Department of Informatics

A Cloud Storage Overlay to Aggregate Heterogeneous Cloud Services

Dissertation submitted to the
Faculty of Business, Economics and Informatics
of the University of Zurich

to obtain the degree of
Doktor der Wissenschaften, Dr. sc.
(corresponds to Doctor of Science, Ph.D.)

presented by
Guilherme Sperb Machado
from
Porto Alegre, RS, Brazil

approved in February 2016

at the request of
Prof. Dr. Burkhard Stiller
Prof. Dr. Filip De Turck

The Faculty of Business, Economics and Informatics of the University of Zurich hereby authorizes the printing of this dissertation, without indicating an opinion of the views expressed in the work.

Zurich, February 17, 2016

Chairwoman of the Doctoral Board: Prof. Dr. Elaine M. Huang

Abstract

Technology advances in the last decade, such as mobile phones, audio and video with high quality, and Online Social Networks (OSN), allowed users to produce, share, and consume data amounts as never seen before. This demand created the necessity of storing and retrieving users' content in a fast, secure, and reliable manner.

In order to bridge the gap for modern storage demands, which includes private and sharing purposes, Cloud computing and Peer-to-Peer (P2P) technologies emerged. The technology employed by Cloud providers is highly elastic, distributed to a high number of servers, geographically spread in different domains, all being interconnected with complex networks. However, from the user's perspective, Cloud providers and their Cloud services are centralized entities, since users have to entirely trust their non-transparent internal process, e.g., users are not aware of what kind of data redundancy and security measures are implemented, nor where their data is stored. Moreover, Cloud services offered by different providers present heterogeneous characteristics: they offer different Application Programming Interfaces (API), accounting and charging schemes, privacy and security levels, functionality, and data type restrictions of files. Despite such heterogeneity, and showing one common aspect among Cloud services, data is stored on Cloud services' servers independently of these various differences.

In contrast to Cloud Computing, Peer-to-Peer (P2P) data storage systems present a higher transparency from the user's perspective: it is known where user's data are stored (the routing and lookup algorithm are known), how it is secured, and under which conditions data will be available or unavailable. However, for data storage purposes, P2P systems also show drawbacks due to churn and lack of a strong identity, leading to a non-suitable storage solution due to its lacking reliability.

Therefore, this thesis proposes an overlay to aggregate Cloud services' storage following a hybrid approach, using centralized and decentralized entities, for upload, download, and sharing files, either publicly or privately. The overlay is referred as *PiCsMu*, the "Platform-independent Cloud Storage System for Multiple Usage". Such overlay is responsible to decide how and where to store user's data, as well to manage the generated metadata by keeping it in a central server (for private files) or in a P2P network (for shared files). The underlay is formed by different Cloud services which stores actual data. The hybrid approach used in PiCsMu takes advantage of the wide storage elasticity of Cloud services, the scalability and privacy of a P2P system, and the stability, reliability, and security of centralized entities.

PiCsMu stimulated three areas of research. First, in order to tackle the heterogeneity problem of storage in Cloud services, this thesis investigates the *data validation* process in Cloud services, to understand how the acceptance of specific file formats is handled for storage purposes. In turn, data can be adapted accordingly to be stored in almost any Cloud service – even those which present data format restrictions –, thus, enabling the aggregation of Cloud services' storage. Second, this thesis defines the system architecture and related processes for upload/download in case of private or sharing purposes. These processes employ data redundancy techniques, encryption schemes, and the placement of data fragments in different Cloud services. Third, this thesis explores the integration of existing OSNs to provide incentives for PiCsMu adoption and to enhance content sharing experience through the means of recommendations. These recommendations are based on data measured from existing OSNs, e.g., measuring which friends are geographically closest to a particular user *and* which of them interact the most with. Such information can give an indication that they might also be interested in adopting the PiCsMu system. In order to achieve that, interactivity- and location-based methods have been developed to measure data collected from existing OSNs.

The PiCsMu system was evaluated as a whole by evaluating the design implementation of its three researched areas. For each of these three research areas the evaluation observed the feasibility, scalability, overhead, functionality, and accuracy of the developed solutions. Additionally, a legal discussion was added, since storage solutions face legal and regulative dimensions upon operation.

The achieved results show that it is possible to build an overlay with Cloud services storage in the underlay, e.g., Google Picasa, Imgur, ImageShack, Amazon S3, and SoundCloud, showing that multiple services with different data format restrictions can be aggregated. Also, these results reveal that PiCsMu scales with respect to different file sizes, showing a moderate data overhead and a low metadata overhead added by the PiCsMu system in case of 1 GByte files. Finally, interactivity- and location-based methods in support of the PiCsMu social recommendations show that they can estimate, respectively, 2 out 5 OSN friends that an OSN user also perceives as he/she interacts most with, and 1.3 out 5 OSN friends that an OSN user also perceives as the geographically closest to.

Kurzfassung

Technologische Errungenschaften des letzten Jahrzehntes, wie beispielsweise Mobiltelefone, hochwertige Audio- und Videoaufnahmen und soziale Netzwerke (OSNs), erlauben Nutzern, niemals zuvor gesehene Datenmengen zu produzieren, zu teilen und zu konsumieren. Hierdurch ergibt sich die Notwendigkeit, Nutzerdaten schnell, sicher und verlässlich zu speichern und abzurufen.

Cloud Computing und Peer-to-Peer (P2P) Technologie etablierten sich, um diesen modernen Speicherbedarf für Privat- und Verteilungszwecke zu decken. Die von Cloud-Anbietern verwendete Technologie ist höchst und über eine Vielzahl von Servern und Domänen verteilt, welche durch komplexe Netzwerke verbunden sind. Dennoch sieht es für die Cloud-Nutzer so aus, als wären Cloud-Anbieter zentralisierte Entitäten, da die Nutzer vollständig den für sie nicht transparenten, internen Prozessen der Cloud-Anbieter vertrauen müssen. So wissen Nutzer beispielsweise oft weder, wie Datenredundanz oder Sicherheit implementiert werden, noch, an welchem Ort die Daten gespeichert sind. Sogar trotz Dienstgütevereinbarungen haben Cloud-Nutzer nach wie vor Risiken zu tragen, beispielsweise bezüglich Datenverlust, Datenlecks und Daten-Nichtverfügbarkeit, wenn sie sich auf einen einzelnen Cloud-Anbieter verlassen. Darüber hinaus haben Cloud-Dienste verschiedener Anbieter unterschiedliche Charakteristika: Sie sind durch unterschiedliche Programmierschnittstellen, Verrechnungs- und Bepreisungsschemata, Privatheits- und Sicherheitsstufen und Funktionalitäten und Einschränkungen bzgl. zulässiger Datentypen gekennzeichnet. Trotz dieser Heterogenität haben alle Cloud-Dienste gemein, dass die Daten auf den Servern des Cloud-Anbieters gespeichert werden.

Im Gegensatz zu Cloud Computing Systemen sind Peer-to-Peer (P2P) basierte Datenspeichersysteme dem Nutzer gegenüber transparenter: Es ist

bekannt, wo die Nutzerdaten gespeichert sind (der Routing- und Lookup-Algorithmus sind bekannt), wie sie gesichert sind und unter welchen Umständen die Daten verfügbar oder nicht verfügbar sind. Wegen des Churn (Peers, die das System verlassen) und des Fehlens starker Identitäten haben P2P-Systeme dennoch Nachteile für die Datenspeicherung, und letztendlich macht sie fehlende Verlässlichkeit als Speicherlösung ungeeignet. Aus diesem Grund entwickelt diese Dissertation ein Overlay, um den Speicherplatz von Cloud-Diensten mittels eines hybriden Ansatzes zu aggregieren, sodass zentralisierte und dezentralisierte Systemkomponenten für Upload, Download, und das öffentliche oder private Teilen von Daten verwendet werden. Dieses Overlay wird als *PiCsMu*, das "Platform-independent Cloud Storage System for Multiple Usage", bezeichnet. Dieses Overlay entscheidet wie und wo die Nutzer-Daten gespeichert werden und verwaltet die dabei anfallenden Metadaten auf einem zentralen Server (bei privaten Daten) oder in einem P2P-Netzwerk (bei geteilten Daten). Das Underlay setzt sich aus verschiedenen Cloud-Diensten zusammen und speichert die tatsächlichen Daten. Der von PiCsMu verwendete hybride Ansatz nutzt die Vorteile der grossen Speicherflexibilität von Cloud-Diensten, die Skalierbarkeit und Privatheit von P2P-Systemen, und die Stabilität, Verlässlichkeit und Sicherheit von zentralisierten Systemkomponenten.

PiCsMu forschte auf den folgenden drei Gebieten. Erstens untersucht diese Dissertation den Prozess der *Datenvalidierung* von Cloud-Diensten, um zu verstehen, wie die Abnahme von spezifischen Datenformaten gehandhabt wird und somit das Problem der Heterogenität von Speichermöglichkeiten verschiedener Cloud-Dienste zu lösen ist. Dieses ermöglicht die Aggregation von Cloud-Diensten, da Daten entsprechend angepasst werden können, um sie dann bei fast jedem Cloud-Dienst abzuspeichern – sogar bei solchen, die Einschränkungen bezüglich zulässiger Datentypen haben. Zweitens definiert diese Dissertation die Systemarchitektur und damit zusammenhängende Prozesse für den Upload und Download von privaten oder geteilten Dateien. Diese Prozesse verwenden Datenredundanztechniken, Verschlüsselungsschemata und das Verteilen von Datenfragmenten über verschiedene Cloud-Dienste. Drittens untersucht diese Dissertation die Integration von existierenden OSNs, um Anreize für die Annahme von PiCsMu zu schaffen und das Erlebnis beim Teilen von Daten durch Empfehlungen zu verbessern. So wird zum Beispiel in existierenden OSNs gemessen, welche Freunde einem PiCsMu-Nutzer geographisch am

nächsten sind *und* mit welchen von diesen er am meisten interagiert, was ein Indiz dafür ist, dass diese Freunde ebenfalls Interesse an des Benutzung des PiCsMu-Systems haben. Hierzu wurden interaktivitätsabhängige und ortsabhängige Methoden entwickelt, mit welchen in existierenden OSNs gesammelte Daten ausgewertet werden können.

Das Design und die Implementierung des PiCsMu-Systems wurden ganzheitlich im Kontext der drei untersuchten Gebiete evaluiert. Für jedes der Gebiete wurde die Durchführbarkeit, Skalierbarkeit, Overhead, Funktionalität und Genauigkeit der Lösungen evaluiert. Da Speicherlösungen gesetzliche und regulative Hürden zu nehmen haben, beinhaltet diese Dissertation eine entsprechende Diskussion der Gesetzeslage.

Die erzielten Ergebnisse zeigen, dass es möglich ist, ein Overlay mit Cloud-Diensten, wie z.B. Google Picasa, Imgur, ImageShack, Amazon S3 und SoundCloud, als Underlay zu entwerfen. Dieses wiederum belegt, dass Cloud-Dienste mit verschiedenen Einschränkungen bzgl. zulässiger Datentypen aggregiert werden können. Es wird bewiesen, dass PiCsMu mit der Dateigrösse skaliert, sodass das PiCsMu-System einen moderaten Daten-Overhead und niedrigen Metadaten-Overhead für 1 GByte grosse Dateien hat. Abschliessend wird gezeigt, dass die interaktivitätsabhängigen und ortsabhängigen Methoden, welche das PiCsMu-System unterstützen, 2 von 5 OSN-Freunden, mit denen der Nutzer glaubt am meisten zu interagieren und 1,3 von 5 OSN-Freunden, die der Nutzer geographisch am nächsten glaubt, prognostizieren können.

Contents

1 Introduction

Technology advances over the last decades allowed individuals to produce data amounts as never seen before as well as sharing and consuming data from different sources [127]. Digital photo cameras, smart mobile phones, free Internet services, distributed file sharing services, social networks, video on demand, and smart TVs are examples of technologies that triggered the production of large amounts of data from users' perspective. All these mentioned examples – hardware or software – became easily accessible in a large scale, thus, creating the necessity of storing and retrieving the generated content in a fast, secure, and reliable manner.

1.1 Approaches to Data Storage and Data Sharing

In order to provide an umbrella for the storage and analysis of large amounts of data generated by many sources, Cloud computing emerged with the ultimate goal to provide computing as a utility [7]. This fact represented a considerable change on how private and corporate users used to think

about computing resources. One of the biggest changes is the possibility to request resources on demand, providing an illusion of infinite resources to its users [7]. As a result, a wide variety of Cloud services were released by several Cloud providers, e.g., Amazon EC2 [5], Dropbox [29], Google Drive [48], Google Picasa [48], SoundCloud [110], turning Cloud computing into one of the most popular technology to persist and share data.

Although Cloud computing presents architectural decentralized characteristics [125] (e.g., distributed set of nodes to provide services), indirectly it also has centralized characteristics (client/server) when seen from the users' perspective. First, users perceive the Cloud as a single entity that provides and manages the computing infrastructure, interacting with a single Cloud provider. Second, users have a contract (or acknowledged Terms of Service) with a single Cloud provider, ensuring the fulfillment of parameters and service functionality. Third, in some Cloud providers such as Amazon [5] and Google Compute Engine [48], users have the possibility to choose data centers located in different geographic locations, but all belonging to the same Cloud provider.

The use of Cloud computing brings a number of advantages in terms of accessibility, elasticity, and costs, but disadvantages exist due to the aforementioned indirect centralized characteristics. One of the disadvantages is to rely on a single Cloud service when they do not provide a transparent measures of, e.g., data redundancy, availability, security, and migration of data. Therefore, for example, if a Cloud service faces a period of unavailability on its services, users would not be able to retrieve the desired data – which can bring several negative impacts for both parties. A contract, i.e, Service Level Agreements (SLA) [82], must be in place between the parties to minimize such impacts. In this case, an SLA serves as an instrument to ensure, e.g., a minimum percentage of availability and time to recover if fails occur. Costs are usually implied if agreed SLA metrics are not fulfilled. However, even with the use of SLAs, unexpected fails can happen. Another disadvantage is the Cloud service's lock-in [80]. After using services of a specific Cloud provider, it becomes challenging to migrate the whole computing resources to another Cloud service that seems more interesting from an economic or technical perspective. In the data storage

context, a typical example would be the migration of several gigabytes from one Cloud service to another (between different providers). The challenge lies on two distinct facts: (a) the heterogeneity of Application Programming Interfaces (API) offered by Cloud services, and (b) the heterogeneity of file formats accepted by them, e.g., while some Cloud services accept to store data with any format, others might only accept pictures and videos, or documents and sheets. Both (a) and (b) require an extensive analysis before committing to any change of Cloud services.

In contrast to Cloud computing, Peer-to-Peer (P2P) systems employ a decentralized architecture to store and retrieve data. A P2P network is formed by several users running a P2P system, being called as peers. Peers are interconnected and interact among them without a central point of coordination. Therefore, a P2P network presents decentralized characteristics not only on the architecture level (i.e., nodes directly connected to other nodes to provide storage), but also on how users perceive the storage service provided by a P2P network. Data is fragmented to be stored on a diversity of peers and, therefore, one single entity is not responsible for the data as a whole – being the opposite when compared to the Cloud computing model. In this context, contracts among peers to ensure a certain service level is a challenging topic not yet employed by current P2P systems.

The use of P2P in the context of data storage bring advantages in terms of scalability, privacy, and data reliability. P2P networks scale for not having a central node that manages and controls all other peers, thus having direct network traffic between the peers. In terms of privacy, when a peer joins the network it does not require a strong identity, i.e., name, email, or address. It only requires an unique identity on the network (e.g., an username or a random identity) that, depending on the P2P system, can/may change over time. Related to data reliability, P2P systems employ data replication, hashing, and indexing to ensure that (1) parts of the stored data are found in many peers in case of, e.g., network issues or high churn [115], and (2) to ensure that the data was not maliciously modified. However, P2P systems also have disadvantages exactly due to its fully decentralized nature. One of the biggest disadvantages is the lack of a strong identity. This fact

can bring problems related to security since malicious users might create a large number of identities to be the majority and consequently manipulate decisions (e.g., data replication) on the P2P network [123]. Additionally, the existence of churn might impact on the overall performance when retrieving files.

Therefore, this thesis explores the opportunity of having centralized and decentralized entities together – beyond the current state of the art – by proposing an overlay to aggregate several storage services following a hybrid approach, being used to upload, download, and share files, either publicly or privately. The overlay is referred as *PiCsMu*, the acronym for Platform-independent Cloud Storage System for Multiple Usage. The hybrid approach for data storage included in PiCsMu takes advantage of the stability, accessibility, elasticity of Cloud services, the scalability, privacy, and data reliability of a decentralized P2P system, and the stability and security of centralized entities. PiCsMu aggregates Cloud services to store data, with the support of a P2P network to manage metadata of shared files. The centralized entities provide several functionalities that support the management of PiCsMu, e.g., identity management. The whole overlay and the interactions between the several components is referred as the *PiCsMu system*, while the application used by end-users to upload, download, and share files, is referred as the *PiCsMu application*. The PiCsMu system is seen as a single storage user-perceived entity, since users only interact to the PiCsMu application in order to upload, download, and share files – even if storage is provided by multiple heterogeneous storage services.

The PiCsMu system is composed by several entities and software components, but three research elements are highlighted in Figure 1.1. The boxes with a vertical pattern highlight aspects focused in this thesis, while the box with horizontal pattern represents the implementation of entities that support the main concepts. In order to rely on multiple Cloud services that offer storage and, thus, tackling the general problem of heterogeneity of Cloud services, this thesis first discusses how data can be stored in any Cloud service despite of the data format that is accepted. The goal is to not have a differentiation between different Cloud services related to storage, but forming a pool of Cloud services that can simply store bytes. This is

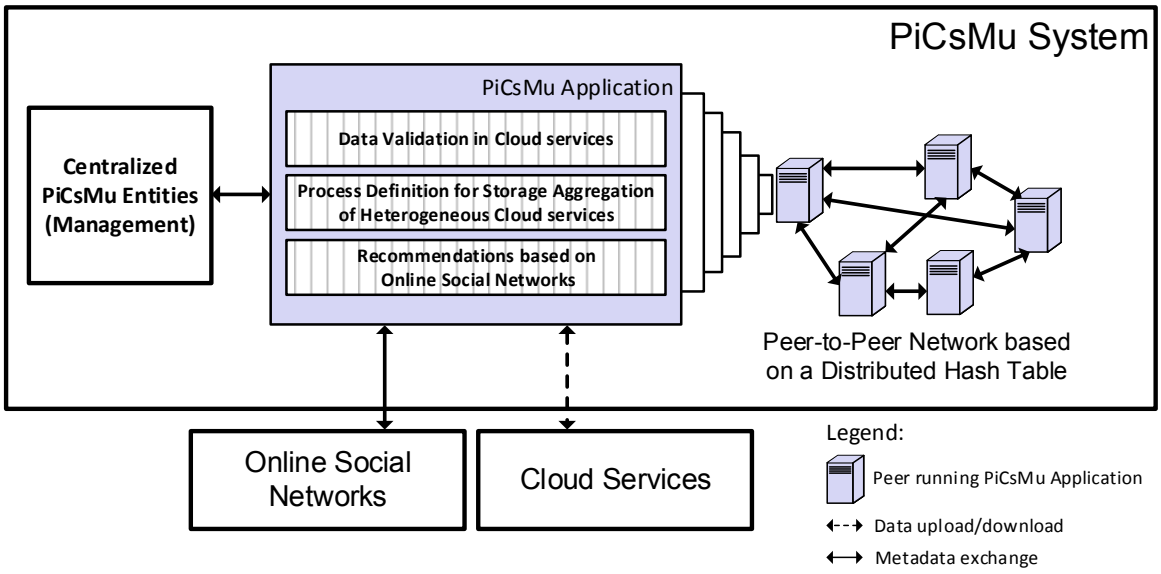

Figure 1.1: PiCsMu system and high-level entities

accomplished by analysing the *data validation* process in Cloud services, understanding how the acceptance of only specific file formats is modelled and implemented.

This thesis defines an overlay, the PiCsMu system with an application, that enables the *aggregation of heterogeneous Cloud services* to store data. The PiCsMu application follows a set of steps that is performed every time a file is uploaded and downloaded, for private or for sharing purposes. Data redundancy techniques are applied to provide data reliability in case of, e.g., Cloud services failure, the loss of Cloud services' credentials, or if malicious users modify fragments of the persisted data. Privacy and security are considered through the use of encryption, ensuring that only authorized users have legible access to stored/shared files. File's data are placed following the rules of a scheduler, which decides how data should be fragmented and where data fragments should be stored. The scheduler's decision is based on factors like, e.g., file size, file format, available Cloud services, and randomness to spread data.

Currently, several applications and systems integrate existing Online Social Networks (OSNs) to the core of their solutions. E.g., Instagram [58] integrates Facebook and Twitter, which enables an easy sharing and spreading of content to these OSNs. The integration is beneficial given that most of these applications do not create their own OSNs, but benefit from exist-

ing OSNs by exploiting their social graph to infer information [92]. This thesis also explores the integration of existing OSNs to provide incentives for PiCsMu adoption and enhance content-sharing experience through the means of recommendations. E.g., measuring which friends that are geographically closest to a particular user *and* knowing which of them interact the most with, give an indication that they might also be interested on adopting the PiCsMu system. Thus, although the PiCsMu application does not have a recommendation system implementation, the last part of this thesis focus on obtaining accurate *interactivity* and *geographical closeness* OSN measurements to serve as an input to a future PiCsMu recommendation system.

1.2 Data Validation in Cloud Services

Cloud services can be classified into three different types [7] Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS), and Software-as-a-Service (SaaS). SaaS is the highest level of Cloud computing, meaning that users do not need to be concerned on PaaS or IaaS in order to use software that runs in the Cloud. SaaS features a complete application offered as a service on demand.

One of the main fundamental SaaS characteristics is to be *application-specific*. As the name says, it is software made available as a service, with a specific purpose and use. Therefore, depending on the application itself, Cloud Providers restrict data pushed into their servers. Google Picasa [48] is an example to illustrate SaaS restrictions related to what is pushed to the application. Within Picasa, nowadays, the user can upload photos but under certain limitations: just a set of image formats (e.g., JPG, PNG, BMP), resolution up to 2048x2048 pixels (under free service plan), and file size up to 20 MByte. In order to enable application-specific restrictions, Cloud services must implement a data validation process [135] to verify what is being uploaded to the SaaS application.

On one hand, Cloud services supporting any data types are termed *generic Cloud storage services*. On the other hand, Cloud services that sup-

port storage of a restricted set of data types – what typically SaaS offers – are termed *data-specific Cloud storage services*.

Hence, this thesis investigates, evaluates (by means of tests), and discusses the data validation process of Cloud services in the scope of *data-specific Cloud storage services* [81]. The value and innovation of this research relies on answering the following research questions:

Research Question 1.1. Is it possible to bypass the data validation process to store arbitrary data into Cloud services' servers?

Research Question 1.2. If it is possible (based on the test results), are there impacts related to *security* as well as *accounting and charging*? These two dimensions are key to Cloud services since reputation and revenue can be affected.

In the scope of *security*, a poor or incorrect data validation process can lead to security vulnerabilities, depending on how the SaaS application is implemented and internally architected – even though this thesis does not aim to expose application-specific vulnerabilities. Also, an input not recognized by the data validation process can allow data injection to Cloud services' servers bypassing content rules and persisting content that the provider is unaware of.

Related to *accounting and charging*, the data validation process must be aware of consistency. First, it must assure that what is declared is actually used (or vice versa). E.g., if an uploaded photo has resolution declared as 80x60 in the header, but the payload data size as 1000x1000 pixels, the image should not be valid. Second, it should be aligned to what the Accounting System (AS) takes into consideration. E.g., if a Cloud service does not account and charge based on storage, the data validation process should verify whether pushed data do not contain an abusive data amount considering allowed file formats. Thus, depending on how Cloud services implement the accounting of resources, a poor data validation process may impact to what users consume (accounting) and what they will have to pay for (charging). The discussion part of this research contribution focuses on: how malicious users can benefit of encoding data inside data (bypass-

ing the data validation an content rules), how Cloud services can have economic losses (accounting/charging), and how Cloud services can be legally affected.

In order to investigate and evaluate the data validation process, a set of encoders and test cases were implemented and performed on well-known public Cloud services (e.g., Google Picasa, Facebook, SoundCloud, TwitPic). The tests simulate a user uploading and retrieving data (i.e., files) via a proof of concept software that constructs files in a particular manner based on different encoders. Each encoder implements a different technique, e.g., using steganography [64], or exploring the use of specific header fields.

1.3 Aggregation of Heterogeneous Cloud Services' Storage

Both types of Cloud services – with *generic* and *data-specific* storage – show one common aspect: data is stored on Cloud services' servers. However, both types of Cloud services are still heterogeneous: they offer different APIs, different accounting and charging schemes, different privacy and security levels, different functionality, and, among data-specific Cloud storage services, they present different data type restrictions. Therefore, the heterogeneity between Cloud services turns the task of aggregating Cloud services' storage (into one single storage entity) a challenging task [74]. Aggregating multiple Cloud services allow end-users to have more integrated storage space, a single entry point to store data, and, enhanced privacy and data reliability.

Hence, this thesis presents and evaluates a novel approach on the area of Cloud storage overlays, consolidated in the Platform-independent Cloud Storage System for Multiple Usage (PiCsMu) [74]. PiCsMu stores data on heterogeneous Cloud services independent of its Cloud service models, i.e., IaaS, PaaS, or SaaS, (a) by aggregating multiple Cloud services' data storage capabilities to be seen as one single storage entity – knowing how data validation is performed in Cloud services, (b) by providing enhanced privacy levels, and (c) by enabling a distributed file sharing network relying on Cloud services' storage instead of peers' storage. Even though the aggrega-

tion of Cloud services' storage within an overlay may loose specific Cloud services' features (like, e.g., rotating a picture within Google Picasa or version control within Dropbox), PiCsMu brings specific storage advantages as (a), (b), and (c).

PiCsMu employs the following 4 steps to aggregate Cloud services' storage and upload files: (1) fragmentation, (2) encryption, (3) data encoding, and (4) data upload to Cloud services. Each of these steps requires a minimal overhead with respect to processing time and storage. These steps, combined, add another layer of tasks to reconstruct original files, therefore turning it even harder for an attacker (or, in case of data leakage, non-authorized users) to gain access to the stored content. The fragmentation step brings advantages in data reliability and redundancy, since multiple fragments can be stored in multiple Cloud services, thus, preventing data loss if a single Cloud service shuts down its services (e.g., Megaupload [130]) or if a Cloud service presents serious availability problems.

The resulting information related to all 4 steps forms the *PiCsMu Index*, which represents *what*, *how*, and *where* data was stored. PiCsMu uses the hybrid approach to store this index either centrally, for private storage, or in a Peer-to-peer (P2P) network enabled by a Distributed Hash Table (DHT), for sharing purposes [77].

The evaluation focuses on measurements related to the file upload and download times, scalability considering different file sizes and amount of users, and system's overhead. Although PiCsMu supports public Cloud services, the evaluation uses a local Cloud service within a local network in order to perform measurements considering the best case scenario (excluding external factors, e.g., delays, which are not part of PiCsMu).

Therefore, this thesis also addresses the following research questions:

Research Question 1.3. Is it possible to build the PiCsMu system and application, exploring *generic* and *data-specific* storage of Cloud services in order to store, retrieve, and share any kind of files?

Research Question 1.4. Would the PiCsMu system scale with respect to different files (with different sizes) being stored, retrieved, and shared?

Research Question 1.5. How much data and metadata overhead is required to exploit jointly *generic* and *data-specific* storage of Cloud services considering data validation in Cloud providers?

1.4 Recommendations based on Interactivity and Geographical Closeness of OSN Friends

An important advantage of integrating OSNs to an application is to collect OSN-specific information that can be used as input to a Social Recommendation System (SRS). For example, an application can use an SRS to: (1) expand the user base, i.e., attract more users to use the application or system, and build its own social graph; and (2) enhance the user's experience, e.g., recommend to share content to specific friends that might be interested, thus, producing a personalized experience for each user. Therefore, an SRS requires to obtain and keep monitoring a variety of information within OSNs (i.e., OSN data), including, e.g., public/private posts and messages, locality tags, user's status updates, user's preferences, and content similarity.

This thesis designs, implements, and evaluates JSocialLib [76], a meta-API (Application Programming Interface) library to collect OSN data from existing OSNs and to provide two methods in support of social recommendations: (1) the interaction- and (2) the location-based methods. The interaction-based method measures the interactivity between OSN users. Thus, e.g., PiCsMu SRS can recommend content based on what OSN friends that most interact with a certain OSN user are consuming. The location-based method measures how geographically close OSN users are to his/her OSN friends. Therefore, e.g., PiCsMu SRSs can recommend new OSN friends being geographically close and using the same application. These methods are complementary and can be used in combination to determine to whom content, items, or new friends should be recommended to. JSocialLib is a meta-API, since it relies on other libraries underneath, such as RestFB [104] and Twitter4J [121], in order to communicate with multiple existing OSNs.

The collection and monitoring of relevant OSN data by third-party applications are challenging tasks, since OSNs (a) impose rate restrictions to their API calls (e.g., Facebook limits 600 calls per 10 minutes per access token), (b) do not provide detailed information about specific OSN features (e.g., how many messages *user A* exchanged with *user B* using a mobile phone, in the last month), and (c) may provide incomplete or not up-to-date data [38]. After collecting OSN data, SRSs must perform a reasoning process with the OSN data, which is also challenging, since it must deal with the lack of data and still present accurate recommendation results: the higher the accuracy is, the higher are the chances that recommendations are accepted by users. This subjective accuracy of recommendations is an important factor to produce attractive recommendations results for users, leading to the following research question:

Research Question 1.6. How accurate can JSocialLib's interaction- and location-based methods be compared to OSN users' perception?

In order to calibrate and evaluate these new JSocialLib interaction- and location-based methods, a study was carried out [76]. The study is composed out of three parts: first, the data set is obtained by a Web-based survey, which indicates the friends that OSN users most interact with and which are the geographically closest to; second, interaction- and location-based methods are calibrated for a random half of the data set collected; third, the calibrated social recommendation methods are evaluated against the remaining half of the data set.

1.5 Thesis Contributions

Motivated by the above observations, this thesis makes the following contributions to the field of Cloud management and storage overlays:

1. introducing and defining a novel problem in the Cloud computing community, by investigating, evaluating, and discussing the *data validation process* of data-specific Cloud storage services, showing the dangerous possibilities on how to bypass the data validation process

to store arbitrary data, ultimately observing impacts related to security as well as to accounting and charging;

2. designing, implementing, and evaluating a Cloud storage overlay (PiCsMu), capable of *aggregating storage space of heterogeneous Cloud services* to be seen as one single storage entity, enabling a distributed file sharing network based on P2P technology and relying on Cloud services' storage instead of peers' storage;

3. designing, implementing, and evaluating a library to collect OSN data from existing OSNs, providing the *interaction- and location-based methods* to serve as a support input to social recommendation systems, respectively measuring the interactivity between OSN users and how geographically close OSN users are to his/her OSN friends, and ultimately generating results that are closely related to users' perception;

4. developing a *PiCsMu* implementation for overall instantiation and validation of the main concepts of this thesis in a practical manner, allowing the presentation of the whole solution in demonstrations.

1.6 Thesis Outline

The remainder of this thesis is organized as follows. Chapter 2 presents related work on concepts that form the foundation upon which this thesis stands. It includes the terminology used throughout the remaining chapters, particularities of Cloud storage services and overlays, the characteristics of file sharing systems based on P2P technologies, as well as related work about social recommendations based on Online Social Networks' data.

Chapter 3 defines the PiCsMu architecture, the processes to upload and download files, the details of each system's building block, as well as the prototypical implementation of PiCsMu, which uses centralized and decentralized entities. One of the key aspects to upload and download files in PiCsMu is how data is represented to store in Cloud services. Thus, Chap-

ter 4 presents an investigation of Cloud services' data validation processes, analyzing how data is stored in Cloud service's servers. Such investigation is composed of a methodology to test Cloud services' data validation rules, using crafted data encoders (developed in this thesis) that are part of the PiCsMu system.

In order to explore Online Social Networks' data to provide incentives to use PiCsMu, Chapter 5 presents JSocialLib, a library that supports Social Recommendation Systems (SRS) in the task of generating recommendations. The JSocialLib design is shown, also its implementation, API calls, and the existing Online Social Networks that are supported.

Since the PiCsMu system combines centralized and decentralized entities, interplaying with several components either in the under- or in the overlay, it is necessary to evaluate PiCsMu as a whole, as well as the different solutions that composes the system. Chapter 6 presents a section for each of the solutions described in this thesis following a bottom-up approach, based on Chapters 3, 4, and 5, respectively. It first evaluates the parts which forms the base for the PiCsMu system, and, afterwards, evaluates the system as a whole. Each of the sections focus on experiments to evaluate the feasibility, scalability, overhead, functionality, and/or accuracy of the solutions that compose PiCsMu. Additionally, a legal discussion is presented as part of the evaluation, since storage solutions face legal and regulative dimensions upon operation.

Finally, Chapter 7 concludes this thesis, summarizing contributions and key findings, also suggesting future work.

2

Terminology and Related Work

In order to compare existing approaches to the work performed in this thesis, fundamental terms are defined initially. Furthermore, selected related work is divided into groups, each presented in a section, having the following structure: summary, comparison, and discussion.

The related work addresses major facets of this thesis' contribution: (a) Cloud storage services, (b) Cloud storage overlays, (c) Cloud services' data validation, (d) P2P file sharing systems, and (e) social recommendation systems. Due to the hybrid approach of applying centralized and decentralized entities file sharing systems are investigated only in the context of major functionality.

2.1 Terminology

Cloud computing is a term to define resources delivered as services (*Cloud services*) over the Internet, including the hardware and software in data centers that provide those services [7]. Among several definitions and classifi-

cation proposals, e.g., [7], [88], and [69], the NIST definition [84] is the one that comprises the most complete and differentiated set of enablers, characteristics, service models, and deployment models. Therefore, the NIST's cloud computing definition is considered throughout this thesis:

Definition 2.1. *"Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This Cloud model is composed of five essential characteristics, three service models, and four deployment models." [84]*

Definition 2.2. *Service models are high-level abstract terms to describe how services are delivered to cloud customers. Within the Cloud model, there are three basic service models: Software-as-a-Service (SaaS), Platform-as-a-Service (PaaS), and Infrastructure-as-a-Service (IaaS) [84].*

SaaS is the highest level of Cloud computing, meaning that users do not need to be concerned on PaaS or IaaS in order to use software that runs on the cloud. SaaS features a complete application offered as a service on demand. An example of this service model is online alternatives to local office applications, like text processors. PaaS is a service that provides the software platform where systems run on. The management of hardware resources demanded by the execution of services is transparent. One of the most well-known examples of PaaS is the Google Apps Engine [48], since it provides a well-defined API in a specific programming language to build scalable applications. Finally, IaaS is the lowest level of Cloud computing and the one that represents the most immediate impact for enterprises [45]. IaaS is centered on computation capabilities, which makes it possible to customize infrastructures from the lowest to the highest level at a low cost.

Additionally, other service models were already discussed in literature: Storage-as-a-Service [126], Metal-as-a-Service [122], Network-as-a-Service [42], Policy Management-as-a-Service [117], Testing-as-a-Service [136], Monitoring-as-a-Service [85], and Wireless Sensor Networks-as-a-Service [26] are some of the proposed models making use of resources as utility.

Definition 2.3. *Cloud service is the term used to represent an specific service delivered as an specific service model.*

Definition 2.4. *Cloud providers are entities (commercial or not) that offer Cloud services.*

Definition 2.5. *A Peer-to-Peer (P2P) system is a self-organizing system with autonomous entities called peers acting in the same manner, i.e., running homogeneous algorithm's implementation. The fundamental characteristic of a P2P system is to employ a shared usage of distributed resources in a networked environment avoiding central services [114].*

P2P systems are based on a distributed network architecture that allows, e.g., distribution of bandwidth usage to reduce bottlenecks while being more robust related to failures.

Definition 2.6. *An overlay network (or overlay in short) is a virtual or logical network on top of another network with addressable endpoints [72].*

Overlays are often used to provide a routing topology not available in the underlying network.

Definition 2.7. *A computer file (or just file) is composed of organizational data and content data. The former is represented by headers or control information, usually following a certain standard.*

In more general terms, *organizational data* can also be called *metadata*, since it is "data about *content data*". The raw term *metadata* brings ambiguity due to different fundamental types, and, therefore, the term "organizational data" is used as a synonym to metadata in this thesis. The content-data is the data which the *organizational data* describes. Files are common digital resources that users interact with Cloud services. In this way, files are in the center of the relationship between users and Cloud services, since files are uploaded, download, and shared to other users.

Definition 2.8. *A file format (or file type) is a particular way that information is encoded for storage in a computer file, as files need a way to be represented as bits when stored on a digital storage medium.*

Definition 2.9. *Generic Cloud storage services, or simply generic Cloud services, are Cloud services that allow the upload of any file format or data, not making any distinction between how organizational or content data is structured.*

Cloud services that are classified as *generic Cloud storage services* do not apply data validation rules for file uploads – since it is not necessary to make a distinction between types or characteristics from organizational and content data. In turn, generic Cloud storage services provide storage for any kind of data types. Examples of generic Cloud storage services include Amazon S3 [5], Dropbox [29], Google Drive [48], SkyDrive [86], SpiderOak [112], NXDrive [94], and Wuala [133].

Definition 2.10. *Data-specific Cloud storage services, or simply data-specific Cloud services, are Cloud services that restrict data uploads based on specific file formats or any other characteristic present in the organizational and content data.*

Cloud services that are classified as *data-specific Cloud storage services* apply data validation rules for file uploads. In turn, data-specific Cloud storage services provide storage only for specific data types. Examples of such services are also classified as SaaS due to the *application-specific* characteristic [7], which requires to narrow the use of the application for a specific purpose – e.g., Facebook [39], Google Picasa [48], Google Docs [48], Google GMail [48], SoundCloud [110], Imgur [55], ImageShack [54], and TwitPic [120].

Definition 2.11. *Encoding is the process by which information from a source is converted into bits (considering a file format).*

Possible *encoding* applications include reduction of the file size (e.g., compression) or hiding data inside other file formats to conceal the original content (e.g., steganography) [64].

Definition 2.12. *Decoding is the reverse process of Encoding, converting these bits back into information understandable by a receiver.*

Usually, a *computer file* has a *file name extension* that is a popular, intuitive, and human-readable method to identify a *file format*. It is represented by a *computer file* suffix separated from the file name by a dot symbol (e.g., filename.mp3, filename.jpg). *File name extensions* are also considered a type of *organizational data* or *metadata*.

Encryption and *decryption* are the most basic applications of *cryptography*. Classical cryptography deals with the problem of enabling secret communication over insecure communication media. Modern cryptography is further concerned with construction of information systems that are robust against a variety of malicious attacks [96].

Definition 2.13. *Encryption applies encoding to transform data into a format that cannot be easily understood by unauthorized parties. Decryption reverts the encrypted data to its original state, and, depending on the method used, can only be successfully performed by one or multiple authorized parties [96].*

Definition 2.14. *A credential is the attestation of authority to access a given Cloud service's account.*

The attestation of authority to access a Cloud service's account can be achieved through providing a security handle, e.g., an OAuth [95] token with restricted access (time out) or a username/password pair with full access to that account.

Definition 2.15. *A user is defined as an entity representing a person or individual, who is authenticated and authorized to use a certain application or service.*

There are specializations of the user definition: e.g., the term *OSN user* is used when a user is uniquely associated to an OSN; *Cloud service user* is a user with a credential in a Cloud service; and *PiCsMu user* is used when a user has a PiCsMu credential.

A *friend* is the counterpart of a positive social relationship, possibly involving trust, intimacy, or acquaintance. When this thesis refers to the term *friend*, it is solely in the scope of an OSN. A *friend* is also a *user*.

Definition 2.16. *OSN friend is used to refer to an OSN user, who is associated to another OSN user through an OSN relationship, which can vary depending on the concepts utilized within each OSN.*

Examples of different types of OSN relationships are, e.g., Facebook which has bidirectional friendship relations, and Twitter that has the concept of unidirectional friendship relations.

The general definition of *interaction* between two human beings encapsulates individuals' action toward another, e.g., to talk, to look, or to smile. However, this thesis focuses on *OSN interactions* (*interactions* in short).

Definition 2.17. *OSN interactions are interactions happening within an OSN triggered by OSN users, depending on intrinsic OSN features or concepts.*

Examples of intrinsic OSN features or concepts include, e.g., Facebook that has the concept to "like" an item, or LinkedIn that has the concept of "endorsing" skills, which are specific.

2.2 Cloud Storage Services and Cloud Storage Overlays

A summary of key characteristics in relation to generic and data-specific Cloud storage services is shown in Table 2.1. "Client-side Encryption" refers to the encryption done by the end-user application, rather than manually by the end-user or by Cloud service's servers. SpiderOak [112], NX-Drive [94], and Wuala [133] provide built-in client-side encryption, while Amazon S3, Dropbox, and the others generic Cloud storage services provide server-side encryption [4], mainly used for when transferring data between their datacenters. Using server-side encryption, the service provider holds the encryption and decryption key pair. However, pure server-side encryption allows the provider to view and access all files, and thus, only provides privacy and security when servers are compromised but without compromising the keys. Using built-in client-side encryption all data is encrypted on the end-user machine, before sending it to the Cloud service – which provides a higher security and privacy levels than server-side encryption only. The disadvantage is that if the user loses the password, the

data cannot be accessed anymore. "Client-side Data Reliability" refers to techniques (e.g., data replication, Reed Solomon) done in the end-user application to provide some level of data reliability in addition to server-side data reliability procedures. All observed generic and data-specific Cloud storage services only provide server-side data reliability, meaning that end-users should trust the guarantees provided by the Cloud service. "Support Multiple Clouds" represents whether a Cloud service supports the aggregation of other Cloud services' storage. Thus, all observed generic and data-specific Cloud storage services do not provide means to expand its storage capabilities with the use of external storage capabilities.

Otixo [98], MultCloud [89], CloudHQ [22], CloudKafé [23], and JoliCloud [60] are examples of Cloud storage overlays, as they present an overlay represented as an application supporting multiple generic and/or data-specific Cloud storage services in the underlay. A Cloud storage overlay is also a Cloud storage service when seen from end-user's perspective, since upload or download are requested directly to the overlay instead to each Cloud storage service. The key difference of Cloud storage overlays to Cloud storage services is that overlays store end-user's data in third-party servers, making decisions where to store based on policies (storage management).

Table 2.2 summarizes key characteristics of Cloud storage overlays. The "Fragmentation" refers to the process of fragmenting files into smaller pieces and distribute them to multiple Cloud storage services. The fragmentation process enables to use several Cloud storage services giving the end-user a larger storage size, also providing data redundancy. The "Client-side Encryption" characteristic has the same meaning as presented in Table 2.1. The "Support Generic Cloud storage services" and "Support Data-specific Cloud storage services" indicate, respectively, that Cloud storage overlays support generic and/or data-specific Cloud storage services in the underlay. The "Transparent Aggregation" characteristic refers to the Cloud storage overlay ability to aggregate either generic or data-specific Cloud storage services in a transparent manner to its end-user. For example, end-users upload any file format and the Cloud storage overlay is responsible to be aware of where such file format, or fragments of it, might be stored or not.

Table 2.1: Characteristic comparison of generic and data-specific Cloud storage services

	Cloud storage service	Characteristic		
		Client-side Encryption	**Client-side Data Reliability**	**Support Multiple Clouds**
Generic	**Amazon S3**	No	No	No
	Dropbox	No	No	No
	Google Drive	No	No	No
	SkyDrive	No	No	No
	SpiderOak	Yes	No	No
	NXDrive	Yes	Yes	No
	Wuala	Yes	No	No
Data-specific	**Facebook**	No	No	No
	Google Picasa	No	No	No
	Google Docs	No	No	No
	Google GMail	No	No	No
	SoundCloud	No	No	No
	Imgur	No	No	No
	ImageShack	No	No	No
	TwitPic	No	No	No
	PiCsMu	Yes	Yes	Yes

Thus, the Cloud storage overlay does leverage to the end-user any responsibility of selecting the Cloud storage service that might be used. The "Data Reliability among Cloud storage services" characteristic similarly refers to the "Client-side Data Reliability" in Table 2.1. The main difference relies on the fact that data reliability techniques implemented by Cloud storage overlays are done by the overlay service itself, which might be on the server-side or in the client-side, depending on the particular service. Moreover, the data reliability in the context of Cloud storage overlays is to prevent that if one Cloud storage service is not available or is shutdown, the overlay

is able to reconstruct the original file based on other sources. The "Centralized Index" and "Distributed Index" characteristics refer to how Cloud storage overlays keep track of where and how files were stored in the underlay: keeping the file indexing in a centralized or decentralized manner.

The patent application entitled "Secure Online Distributed Data Storage Services" [99], filled in four different jurisdictions (United States, European Union, China, Canada), was submitted in January 10th, 2014, and it was published in July 17th, 2014. The patent application describes a system called *Data Vaporizer* that provides distributed data storage, focusing in security using public Cloud services. The *Data Vaporizer* uses fragmentation to multiple storage nodes – i.e., public Cloud services – in order to reduce local disk failures, encryption (generating a different key for each of the data parts), and erasure codes for data redundancy.

The patent application entitled "Virtual File System Integrating Multiple Cloud Storage Services and Operating Method of the Same" [65], hereafter called as *Virtual FS,* filled in one jurisdiction (United States), was submitted in December 10th, 2013, and it was published in June 12th, 2014, claiming priority to a Korean Patent Application Number 10-2012-0142754, filled in Korean Intellectual Property Office (KIPO). The patent application describes a virtual file system which has the goal to integrate and manage several Cloud storages. The virtual file system comprises of several entities to, e.g., keep state information of the Cloud storages, store metadata of Cloud storages, receive user requests, select the appropriate Cloud storage for user's requests, transfer converted open API calls to the selected Cloud storage, among others. Moreover, the patent mentions that the decision on what is the most appropriate Cloud storage to hold user's data is based on a Cloud monitoring and performance analysis (upload/download measurements).

Even though *Data Vaporizer* system [99] and the *Virtual File System* [65] present similarities, both approaches differ in functionalities and operation modes to other solutions. Neither [99] or [65] deal with heterogeneity among different existing Cloud services. In contrast to [99], the patent [65] does not explicitly mention the idea of fragmenting data and upload

them to different Cloud services, nor the use of encryption for each fragmented data portions.

Table 2.2: Characteristic comparison of Cloud storage overlays

Characteristic	Cloud Storage Overlay							
	Otixo	**MultiCloud**	**CloudHQ**	**CloudKafé**	**JoliCloud**	**D. Vaporizer** [99]	**Virtual FS** [65]	**PiCsMu**
Fragmentation	No	No	No	No	No	Yes	No	Yes
Client-side Encryption	No	No	No	No	No	Yes	No	Yes
Support Generic Cloud storage services	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Support Data-specific Cloud storage services	Yes	No	No	Yes	No	Yes	Yes	Yes
Transparent Aggregation	No	No	No	No	Yes	No	No	Yes
Data Reliability over multiple Cloud storage services	No	No	No	No	No	Yes	No	Yes
Centralized Index	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Distributed Index	No	No	No	No	No	No	No	Yes

2.3 Data Validation in Cloud Services

Several research works have already explored the data validation process of systems and file formats, but not in the context of Cloud services, as shown

in Table 2.3. For example, solely focusing on security, [15] proposes a technique to automatically generate exploits from software patches that target input data validation vulnerabilities. [135] presents a framework for the validation of data and rules in Knowledge Base systems. For the such systems, the framework defines the validation of data, validation of rules [50], and the interaction between data and rule validation. [43] proposes a data security and data validation framework for a SOA (Service Oriented Architecture) based system. The paper explains what are the employed levels of data security: user authorization access, data encryption, and data validation. As stated in [43], [101] also stresses that the main causes for errors, vulnerabilities, and consequently economically-related impacts, are due to poorly or non-validated data. JSTOR [62] and the Harvard University Library [51] collaborate on a project called JHOVE [111], developing an extensible framework for format validation. JHOVE is an open-source software framework providing functions to perform format-specific identification, validation, and characterization of digital objects (i.e., files). Currently, supported file formats include, e.g., GIF, HTML, JPEG, PDF, TIFF, WAV, and XML. Files are analyzed and checked for being well-formed, i.e., by checking the format standards' requirements.

On the topic of injecting data in Cloud services, GMailFS [129] is a tool that builds a file system using Google GMail service. GMailFS allows the storage of files, independent of their type and size. This is accomplished by segmented email attachments. Even misusing the SaaS application purpose, this tool does not exploit data validation weaknesses. Thus, GMailFS does not use techniques to mask injected content in order to bypass Cloud providers' data validation.

2.4 P2P File Sharing Systems

Although other P2P systems are not considered in the comparison, each selected system represents the characteristics of a typical group of P2P systems. Table 2.4 summarizes the comparison of P2P file sharing systems, where the following dimensions are taken into consideration: "Topology" determines whether the network topology is centralized or decentralized;

Table 2.3: Comparison between related work in the area of system's data validation processes

Characteristic	Related Work in Data Validation of Systems				
	[15]	[135]	[43]	[111]	**This Thesis**
Consider Data-specific Cloud Storage Services	No	No	No	No	Yes
Methodology/Framework for Data Validation Tests	Yes	Yes	Yes	Yes	Yes
Implementation of Data Validation Tests	Yes	Yes	Yes	Yes	Yes
Aware of File Formats' Standards	No	No	No	Yes	Yes
Points Impacts on Security	Yes	No	Yes	No	Yes
Points Impacts on Accounting and Charging	No	No	Yes	No	Yes

"Architecture" illustrates the overlay scheme as structured or unstructured; "Lookup" represents the implemented protocol as being able to query other peers for information; "Efficiency" shows if the content lookup is guaranteed or how efficient it is; "Bootstrapping" describes the mechanism on how a peer joins the network; "Storage on Peers" represents whether peers store file data or not; "File Search" determines how the user can search for files within the P2P network; "Download" determines the entity where data is downloaded from; "Upload" determines the entity where data is uploaded to; "Public Sharing" determines if the system supports to share files

with all the peers on the network; "Private Sharing" determines if the system supports to share files with specific peers only.

The "Topology" dimension is considered centralized when peers act as clients communicating with a central server. BitTorrent is considered both centralized and decentralized since it uses servers acting as trackers, and also a DHT which is completely decentralized. The "Architecture" dimension is considered unstructured when a system does not establish a particular structure on the P2P overlay network, forming such network by nodes that randomly connects to each other. Structured P2P overlay networks, on the other hand, always present the same way of organizing the peer connections. The most common implementation for structured overlays is the DHT. BitTorrent is considered to have a structured and unstructured architecture also due to the use of trackers and DHTs.

Related to the "Lookup" and "Efficiency" dimensions, BitTorrent and PiCsMu use a DHT to find references to content and thus both present efficiency of $\mathcal{O}(logN)$. The only difference is that BitTorrent first perform a lookup to the DHT to find possible trackers, while in PiCsMu there is no notion of trackers – the DHT already presents the references to the content. Napster only needs one query to the central index server to get content information. If the content is available, the lookup is guaranteed to find it. While the central server makes the lookup extremely efficient, it can also lead to failure. If the central index server is down, the whole system becomes inoperable. The same applies for the trackers in BitTorrent. Gnutella is the least efficient solution. Flooding the network generates a lot of unnecessary traffic. Furthermore, the lookup is not guaranteed to find the content, even if it is available somewhere in the network. This happens due to the Time-to-Live (TTL) attribute in each content request. The TTL may expire before a peer holding the content is finally found. Freenet provides a lookup solution that is better than flooding but worse than a DHT. Each peer holds a lookup table to select a most likely optimal peer to forward a message. Since the table is maintained only by the peer itself, it is less powerful and accurate than a DHT. The lookup efficiency is ideally $\mathcal{O}([logN]^2)$, which is unsatisfactorily in comparison with a DHT approach like BitTor-

rent or PiCsMu. As it happens in Gnutella, each message has a TTL and may expire before it succeeds. Therefore, the lookup is not guaranteed.

The most popular bootstrap method is to use a bootstrap server or a set of well-known online peers. In addition, Gnutella caches peers from previous sessions to connect again after a restart. Based on the information contained in the torrent files, or in the DHT, peers in BitTorrent know to which tracker they have to bootstrap. The responsible tracker gives the newly arrived peer the addresses of a set of other peers interested in the same content. This procedure makes the lookup efficient and guaranteed. However, this approach is only viable due to the whole concept of BitTorrent. To summarize, the bootstrap method solely analyzed has no significant influence on the performance of the system. Research in this area already investigated the impacts of different bootstrap methods [20, 132], but the choice mainly depends on good interaction of all components.

Napster, Gnutella, BitTorrent, and FreeNet store file data on the peers, besides maintaining metadata as *meta-files*. Similar to a torrent file, the meta-files contain information on where the file data is actually stored. Napster, Gnutella, FreeNet, and BitTorrent also maintain a global index and connect peers that are interested in the same file. A peer uploads or downloads the file directly from one or more other peers without intermediary entities for data transfer.

All P2P file sharing systems should present a functionality in which users search for files: users type a key term into the application and it retrieves a list of files available in the network. While Napster and PiCsMu can find every file in the whole network that match the search criteria, Gnutella can only return results from the peers it reached during the query. "Internal" file search means that the P2P file sharing system has the file search embedded into the protocol/application, while "External" means that the file search is totally decoupled. BitTorrent and Freenet do not have an internal search function, since the torrent files or file keys have to be searched and obtained through other means, e.g., a website that offers torrent files or file keys. An advantage of the internal file search is to be considered more up-to-date compared to the external. Using an external file search, there is a time gap between the time when the file was made available to the search,

Table 2.4: Comparison between P2P file-sharing systems and PiCsMu

Criteria	Peer-to-Peer File-Sharing System				**PiCsMu**
	Napster	**Gnutella**	**BitTorrent**	**Freenet**	
Topology	Central-ized	Decentral-ized	Centralized/ Decentralized	Decentral-ized	Decentral-ized
Architec-ture	Unstruc-tured	Unstruc-tured	Structured/ Unstructured	Unstruc-tured	Struc-tured
Lookup	Central Index	Flooding	Tracker/ DHT	Key-based	DHT
Efficiency	One-hop	Not Guaranteed	$\log N$	Not Guaranteed	$\log N$
Boot-strapping	Server	Server/ Cache	Server (Tracker)	Known Peers	Server/ Known Peers
Storage on Peers	Yes	Yes	Yes	Yes	No
File Search	Internal	Internal	External	External	Internal
Download	Peers	Peers	Peers	Peers	Cloud Services
Upload	Peers	Peers	Peers	Peers	Cloud Services
Public Sharing	Yes	Yes	Yes	Yes	Yes
Private Sharing	No	No	Yes	No	Yes

and when users actually attempt to download it. During this gap the file might become more popular or, in the worst case, totally disappear from the network. Since the external search is not coupled in any way with the P2P system, the user has no guarantee at all that files would exist in the network. Thus, the chance that files are found in the P2P network with an internal file search is higher.

Although Napster, Gnutella, BitTorrent, FreeNet, and PiCsMu present public sharing functionality, the private sharing among peers is only possible with BitTorrent and PiCsMu. In BitTorrent, private sharing is available since the file contributor has to create its own tracker and adjust the tor-

rent file. Napster and Gnutella have no option to enable private sharing due to their architecture. Freenet does not offer private sharing in its network, however, by following the Darknet approach [21] peers may establish manual direct connections to other peers. Thus, users could build a friend-to-friend network for private sharing – but it is not part of the FreeNet system implementation.

2.5 Social Recommendation based on existing Online Social Networks

Traditional recommendation methods only have one information input type, e.g., "items bought by user A on Web site X" is the input for a recommendation method to provide further recommendations to user A. [138] summarizes traditional recommendation methods into the following categories: content-based [9], collaborative filtering [27], clustering model [14], graph model [1], and association rule graph [90]. Social recommendations are a new approach in the field of traditional recommendation methods [103], since they aim to specifically recommend OSN items and OSN friends to a given OSN user. In contrast to traditional recommendations, social recommendations have to deal with heterogeneous information of one or more sources. The heterogeneity is due to different elements that OSNs introduce, e.g., users, pages, likes, posts, private messages, tags, locations, or complex relationships between all of these items [19]. JSocialLib, the library developed in this thesis to support PiCsMu to recommend OSN items and OSN friends to specific OSN users, belongs to the social recommendation method approach.

JSocialLib applies a modified collaborative filtering approach with implicit OSN data collection [27]. The interaction- and location-based methods do not actively interact with the OSN user in order to infer recommendations. Instead, it uses an implicit OSN data collection to measure OSN interactivity and geographical closeness. Table 2.5 shows a comparison between JSocialLib and relevant works in the area of recommendation methods. The columns "Interaction-" and "Location-aware" indicate whether this work considers OSN interactivity or attached locations in OSN data,

respectively. The column "Evaluation OSN Data Set" determines whether the work uses public or private OSN data to perform the evaluation.

[90] combines a network topology information with genetic algorithms to recommend friends. The work shows that the combination performs better when compared to single traditional recommendation methods. [134] introduces an improved algorithm for personalized collaborative filtering in OSNs, and shows that a higher recommendation acceptance is achieved compared to a traditional collaborative filtering. [17] introduces the concept of user-intimacy through OSN users' interaction and creates a hybrid recommendation method to combine user-interaction, content-based, and collaborative filtering. Another recommendation method based on social interaction was developed in [92], which identifies different types of interactions with different weights. It shows that Facebook users like an item (e.g., posts, videos, or pictures) five times more rather than to comment on it. [138] uses a random walk method and a pair-wise learning algorithm. Experimental results present improvements in recommendation compared to baseline methods. [19] builds a hybrid recommendation, which combines similar interests of OSN users with location similarity. [137] also bases their OSN friend recommendation on location data and social network topology.

[46] developed a method to crawl representative and unbiased samples of Facebook users and is used in most of the related work shown in Table 2.5. JSocialLib does not use [46], since the interaction- and location-based methods rely on OSN data inherent to OSN users' account, and, therefore, do not rely on publicly accessible data that can be constantly crawled. Moreover, JSocialLib uses a combination of public and private OSN data information to conduct the calibration and evaluation of its methods. Therefore, the collection of private OSN users' data differs from other research works.

Different approaches to measure OSN interactivity and geographical closeness were discussed by several authors. [131] proposes the use of interaction graphs to quantify OSN user interactions. It analyzed interaction graphs derived from Facebook and shows that it has significantly lower levels of the "small-world" properties. [124] conducted a measure-

Table 2.5: Comparison of approaches that perform and support social recommendation

	Recommendation Categories	Interaction-aware	Location-aware	OSN Data Set
[90]	genetic	No	No	public
[134]	modified collaborative filtering	No	No	public
[17]	content-based, collaborative filtering	Yes	No	public
[92]	ranking	Yes	No	public
[138]	random-walk, pair-wise learning	No	No	public
[19]	affinity matrices, interest familiarity	No	Yes	public
[137]	random-walk, ranking	No	Yes	public
JSocial Lib	modified collaborative filtering, supports social recommendation, ranking	Yes	Yes	public and private OSN user's data

ment study of the Flickr OSN and shows that only a small fraction of users in the main component of the friendship graph is responsible for the vast majority of user interactions. [71] analyzed the geographical location of LiveJournal users and shows a strong correlation between friendship and geographic proximity. [67] observed users on Twitter and analyzed geographical spread of Twitter usage in relation to users' behavior. [25] analyzed geolocation information of more than 17, 100, and 3.5 million users in Twitter, revealing the presence of four types of country locality user profiles. Differently to what have already been developed, JSocialLib focuses on providing results that should be as close as possible to what OSN users

perceive – related to the perception of the OSN friends that they most interact, and the OSN friends that they are geographically closest.

2.6 Contribution Opportunities and Discussion

The related work approaches presented in this chapter has revealed the following:

1. the lack of investigation about the data validation process in the area of Cloud computing, more specifically on SaaS services, which is important due to the wide range of input possibilities uploaded by end-users and resources consumed;

2. although several P2P file sharing applications and Cloud storage services offer a wide range of storage possibilities to end-users, current approaches do not provide a seamless aggregation of Cloud storage services, only relying on single storage entities instead of multiple ones, and also being dependent of data formats accepted by storage services; and

3. the lack of a tool to support social recommendations in applications (e.g., applications of cloud storage services), possible to integrate and support application's SRSs, being aware of how much OSN users interact to each other and how geographically close they are.

Having identified the above shortcomings, and in direct relationship with the five observations made in Section 1.5, the following opportunities for scientific contribution have been revealed:

1. a novel investigation and discussion of data validation processes from existing SaaS services, proposing a methodology for testing and evaluating Cloud service's data validation, developing a prototype that automatically craft, upload, and download data input (i.e., files), also collecting evaluation results, allowing to observe if it is possible to bypass data validation processes to store arbitrary data, and analyze the impact of results related to security, as well as accounting and charging;

2. the design and implementation of a novel Cloud storage overlay, seen as a single user-perceived entity to upload/download private and shared files, aggregating heterogeneous Cloud services in the underlay, defining upload and download processes to distribute data fragments in multiple Cloud services, being independent of specific file formats accepted by underlay services, providing data reliability through redundancy and error correction algorithms, ultimately being scalable and keeping overhead to a minimum; and

3. the design and development of mechanisms to measure interactivity and geographical closeness of OSN users, providing results closely related to users' perception, being able to support social recommendation systems that are integrated to applications.

Hence, Chapters 4, 3, and 5 present a detailed description of the above contributions.

3

PiCsMu Architecture, Processes, and System

PiCsMu defines how files are handled as well as where file data is stored. In order to achieve it, this chapter presents the design objectives aimed for the PiCsMu system, the system architecture, the information models, and the definitions of file upload and download processes for storing and retrieving files.

The file upload and download processes present three modes of operation: private storage, private sharing, and public sharing. While private storage relies on a centralized index, private and public sharing modes are used with the support of a P2P network. In addition, all three modes of operation are presented in a use case.

3.1 Design Objectives

As described in Chapter 1, Cloud storage overlays require a mechanism to enable and manage the aggregation of several storage entities, scaling to a large number of users storing/retrieving files. User's files should only be accessed by whom is authorized, also presenting a scheme to deal with data losses. Last but not least, the Cloud storage overlay must present a minimum amount of overhead with the add-on functionalities in place. Other requirements could be pointed, such as, e.g., capacity to cache files or monitor Cloud services, but these are faced as performance enhancements, and are not essential for a fully operational Cloud storage overlay with the goal of upload, download, and share files.

Thus, based on these requirements, main design objectives of the PiCsMu system are presented as follows:

1. **Storage aggregation:** Users are able to aggregate several Cloud services into a single user-perceived storage entity for private or sharing purposes;

2. **Scalability:** The system should scale to a high number of users, even when storing large files and sharing/receiving files to/from a high number of receivers/sharers;

3. **Security:** Files should be only accessible to users that are authorized to access them, thus authorization and end-to-end encryption mechanisms shall be employed;

4. **Decentralization for file sharing:** In order to allow users to share/receive files without a large infrastructure and fully benefit from Peer-to-Peer properties, decentralized entities should be present to support the management of data storage – except the Identity Provider, which is designed as a centralized entity due to security reasons;

5. **Data reliability:** The PiCsMu system should provide a mechanism that, even if file fragments cannot be downloaded, the file should become available and without errors; and

6. **Low overhead:** Data encoding techniques, as well as the index design, introduce a considerable amount of additional data stored and messages exchanged – therefore data and network overhead must be low.

3.2 Architecture

Figure 3.1 illustrates the PiCsMu system architecture and its interactions with external entities. The PiCsMu system forms the overlay, and is composed of 6 entities. The *PiCsMu application* (PM-App) is responsible for providing to *PiCsMu users* means to upload, download, and share files, also, providing an interface to create a *PiCsMu identity*. A PiCsMu identity is created before using the PiCsMu application. The *PiCsMu P2P bootstrapping servers* (PM-P2P-Boot) provide means to the PiCsMu application to join the PiCsMu P2P network (PM-P2P). Thus, at least one PiCsMu P2P bootstrapping server has to be known by the PiCsMu application. During a file upload and download processes, the application interacts with overlay and underlay entities. Due to the aggregation of Cloud services, the PiCsMu application interacts with multiple entities in the underlay, which is composed of one or multiple Cloud services. However, the PiCsMu application provides an interface that PiCsMu users perceive it as one single Cloud storage entity, even formed by multiple Cloud storage services.

The *PiCsMu P2P network* and the *PiCsMu central index server* store file index information based on the uploaded files. The index keeps track of files stored using the PiCsMu application, e.g., where the file data is located, and how to access it (cf. Section 3.4.1). The PiCsMu P2P network is maintained to persist index information of shared files. A P2P network was chosen due to scalability and performance metrics, as well as to avoid a single point of failure. The PiCsMu central index server persists index information of private files, i.e., files that are not meant to be shared with other PiCsMu users, but to be kept for private use. A centralized index server was chosen, since a controlled environment represents a more reliable system (in terms of stability and data availability), if compared to P2P networks. Centralized index servers do not share content to others, avoiding index informa-

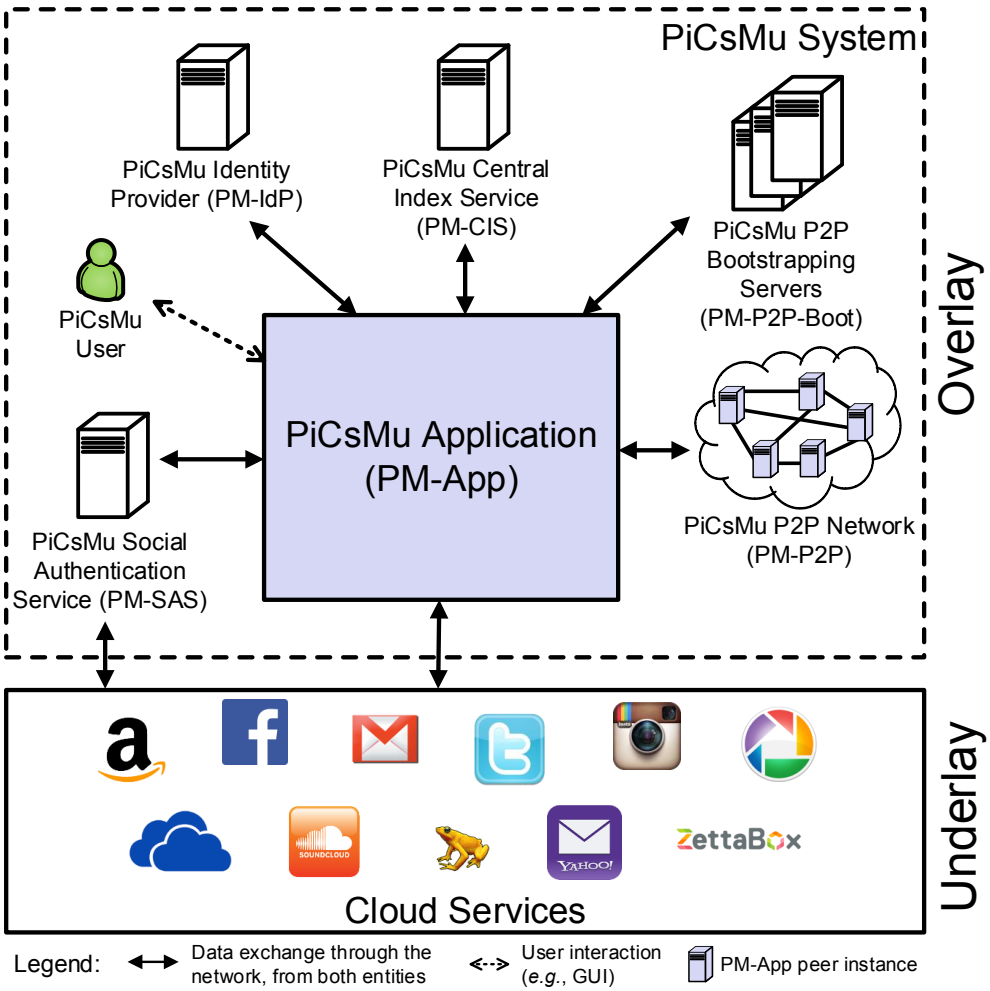

Figure 3.1: Architecture of the PiCsMu system

tion being distributed several times to multiple PiCsMu users, thus, possibly impacting the servers' performance. The underlay, which is not provided by the PiCsMu system, is composed out of one or multiple Cloud services, each associated with a valid credential. While the overlay stores references to data parts, the underlay stores the actual data. For downloading files Cloud services credentials are not necessary, however, to store and/or share files, the PiCsMu user must provide at least one Cloud service's credential beforehand.

A PiCsMu user has to create a PiCsMu identity within a *PiCsMu identity provider* (PiCsMu IdP) using the PiCsMu application. Several PiCsMu IdPs can coexist, but the PiCsMu application is bound to only one PiCsMu IdP. A PiCsMu identity consists of the PiCsMu identifier (PM-ID), the PiCsMu public key, and an encrypted PiCsMu private key. The PM-ID

is a unique identifier (e.g., username) to distinguish PiCsMu users within the PiCsMu system. The public and private keys are generated, when the PiCsMu user creates his/her identity. For the key generation, the PiCsMu application uses RSA, with a key length of 1,024 Byte. The application uses Password-based Encryption (PBE) to encrypt the private key and, therefore, persists the encrypted private key within the PiCsMu IdP. The advantage of encrypting the private key with a password is to prevent the user on maintaining a private key locally (most of times persisted in a hard disk), which can represent a security exposure. However, if the PiCsMu user loses such password, another key pair must be generated, and the PiCsMu IdP should be contacted. Moreover, it is also possible to set pre-existing RSA keys to create a PiCsMu identity.

Besides holding identity information about PiCsMu users, the PiCsMu IdP also keeps online social relations between PiCsMu users. PiCsMu users can have a friendship relation, thus creating the *PiCsMu social network* for file sharing purposes, e.g., users receive share notifications once friends shared a file with them. Moreover, PiCsMu users can associate their PiCsMu identities to an existing OSN and, therefore, the *PiCsMu social authentication service* (PM-SAS) server is used to provide support to specific authentication methods (e.g., password-based, OAuth) in different OSNs. The PM-SAS server receives a request from the PiCsMu application to perform the authentication and authorization of a particular OSN. Such authentication and authorization grant permissions to the PiCsMu application, allowing it to perform actions on behalf of an OSN identity – e.g., obtain a list of OSN friends, check public posts between the OSN user and its friends, and send private messages to OSN friends. Thus, the result of the authentication and authorization by PM-SAS is a token generated by the OSN. The token is then associated to the PiCsMu identity that requested the authorization and authentication – in other words, the owner of the OSN identity. The association between existing OSN identities and a PiCsMu identity is kept in the PiCsMu IdP.

To define the PiCsMu file upload and download processes, respectively, Figure 3.2 is shown. A PiCsMu user selects a file to upload, and chooses its storage mode: for sharing or private purposes, which are described in Section 3.5. The user is responsible for providing valid Cloud services credentials. Once the file and Cloud services credentials are set, the PiCsMu application continues with (1) fragmentation, (2) encryption, (3) data encoding, and (4) data upload to those Cloud services, which are based on the Cloud services credentials the user has provided. Within step (1), files are split generating file parts or file fragments. Each of these file parts are encrypted in (2), resulting in encrypted file parts. Step (3) encodes each of the encrypted file parts using a data encoder, which converts the file part into a specific file format (each file format chosen based on available Cloud services credentials and implementation decisions), resulting in an encoded file part containing encrypted data. E.g., data can be encoded into an image file through the means of steganography [64]. Finally, the file part encoded is uploaded to a Cloud service in step (4), using Cloud services' API.

The information produced in steps (1), (2), (3), and (4) forms the PiCsMu index, which the information model is described in Section 3.4.1. Once the PiCsMu application uploads all encoded files to Cloud services, the application should persist the generated index. While for private storage the PiCsMu system uses the PiCsMu centralized index service to persist private index information, for shared files the PiCsMu application uses the PiCsMu P2P network.

Using the PiCsMu index information, the PiCsMu system is able to retrieve and reconstruct a previously stored file, applying the reverse process: downloading encoded files from one or many Cloud services, decoding these files to obtain encrypted file parts, decrypting them to obtain plain file parts data and, finally, joining all file parts to acquire the original file that was stored.

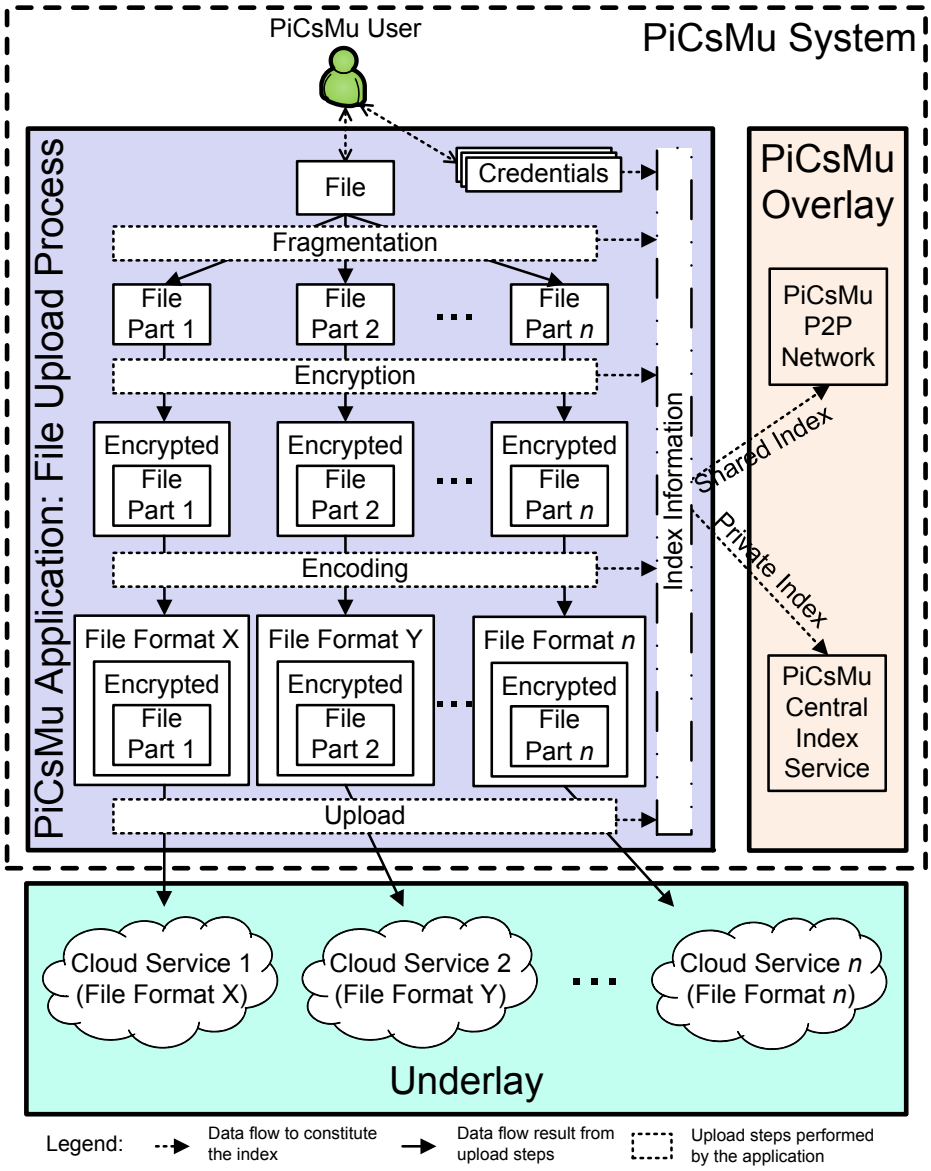

Figure 3.2: File upload process

3.4 Information Model

The PiCsMu system defines three different information models: the index information, called *PiCsMu index*; the share notification information, called *PiCsMu share notification*; and the friendship information, called the *PiCsMu friendship info*. The index and share notification information are responsible to describe the organizational data of an uploaded file and a notification of who shared such file, respectively. The friendship information describes how friend relationships between PiCsMu users are represented.

3.4.1 Index Information

The PiCsMu index is the result of the PiCsMu file upload process, and it consists out of information entities that contains all parameters necessary to locate and reconstruct a file within the PiCsMu system. As a comparison to the design of widely used file systems, e.g., ext4, the PiCsMu index is comparable to the *inode* structure. It carries metadata information about files, and where data is located within a storage media. While PiCsMu index does not present the notion of, e.g., directories or permissions, which are typically present in *inodes*, the design can be extended to include the same features as UNIX-like file systems. The design of PiCsMu index prioritized features to show the feasibility of representing information about files stored specifically using the PiCsMu system.

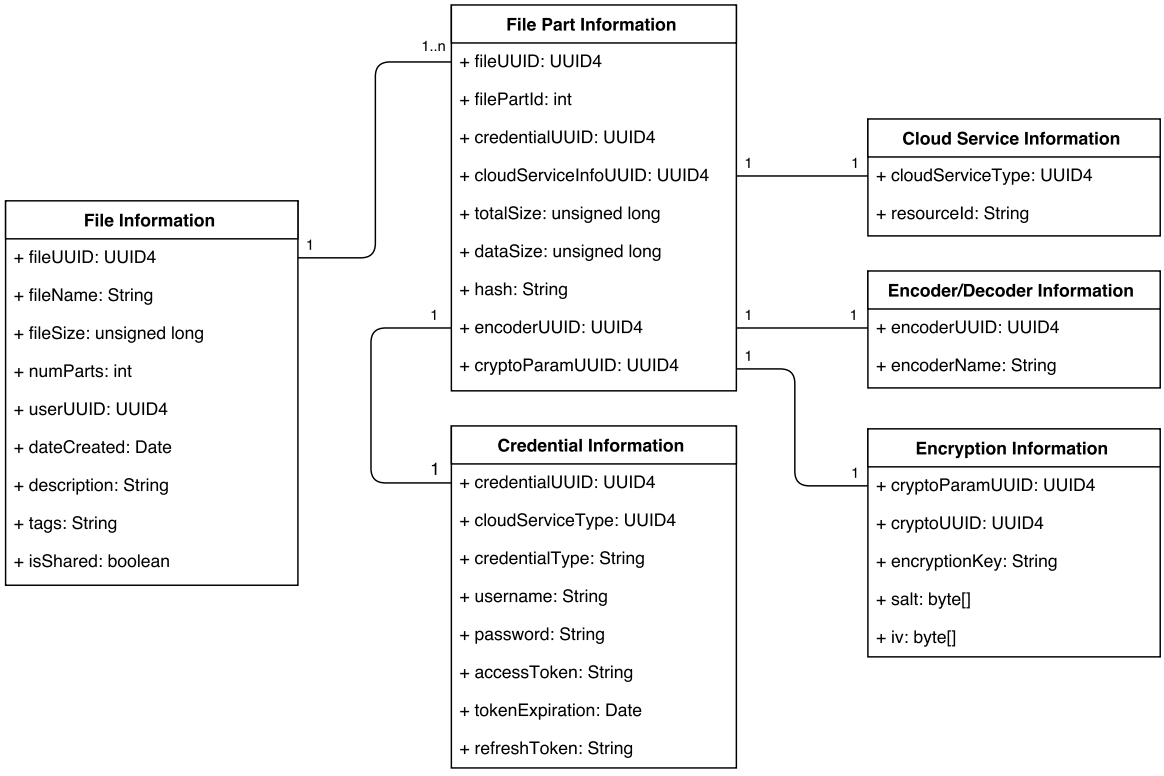

Figure 3.3: Class diagram representation of the PiCsMu index

The PiCsMu index consists of three independent top-level entities: the File Information, Credential Information, and File Part Information. Only with all three entities the PiCsMu application can find all corresponding file parts and can reconstruct the original file. Figure 3.3 depicts the class

diagram of such entities. The description of each class is presented as follows:

File Information: The File Information corresponds to the description of a file in the PiCsMu system. Each file is identified by a Universally Unique Identifier (UUID), using UUID version 4 [68], relying on random numbers. In addition, mandatory information such as file name, file size, upload date, whether it is shared or not, and number of file parts are included. Optional information as description and tags may be included by the PiCsMu user.

Credential Information: The Credential Information corresponds to the Cloud service credential where the encoded file part was stored. It is composed of the unique identifier ("credentialUUID" attribute, represented as a UUID), the credential type ("credentialType" attribute, being possible to set values as "OAuth" or "username/password" pair), the Cloud service that it belongs to ("cloudServiceType" attribute), the credential itself, which can be an OAuth token ("accessToken" attribute) or username/password string ("username" and "password" attributes), the access token expiration time ("tokenExpiration" attribute), and the refresh token ("refreshToken" attribute), which is used to obtain a new accessToken when it expires. Credential Information instances are associated to one or more File Part Information entities through the Credential Information unique identifier.

File Part Information: The File Part Information corresponds to one File Information, and it is mainly composed by a unique identifier, a credential identifier, and a file part size. The file part unique identifier is composed out of the file UUID concatenated with the file part order identifier, starting from "0". Therefore, each file part is considered unique by the PiCsMu system relying on the file UUID information plus the file part order. The credential identifier, which also is a UUID, points to an existing Credential Information entity. The File Part Information instances are associated to one or more File Information instances also through the File Information unique identifier, and, if a File Part Information instance is updated

(e.g., the file part was upload to a different Cloud service due to fault tolerance reasons), it does not require to update the File Information instance. Further elements of the File Part Information are Encryption Information, Encoding/Decoding Information, and Cloud Service Information:

- *Encryption Information:* The Encryption Information is composed out of a salt, an Initialization Vector (IV), and a password ("encryptionKey"). Each file part is encrypted separately, using a PBE method, with a randomly generated IV and password. The "cryptoUUID" attribute represents the unique identifier of a encryption method implementation, and it should be available in the PiCsMu application.

- *Encoding/Decoding Information:* The description of the encoder/decoder used for each encrypted file part. Each encrypted file part is encoded separately, with the encoder being chosen based on which Cloud service the resulting encoded file part will be uploaded to, as well as based on the Cloud service file format restriction. Hence, the used encoding algorithm and the embedded file format have to be placed in the index. The "encoderUUID" attribute represents the unique identifier of a data encoder implementation, and it should be available in the PiCsMu application.

- *Cloud Service Information:* Knowledge of where an encoded file part was stored described as the resource identifier, represented by the "resourceId" attribute. This attribute keeps the specific location (i.e., URL or internal Cloud services' unique identifiers such as, e.g., a picture identifier within a Google Picasa account but not accessible as an URL) where data was stored in Cloud services used by the upload process. The "cloudServiceType" attribute – that is also present in the Credential Information class – represents the unique identifier of the API interface for a specific Cloud service, and it should be available in the PiCsMu application.

3.4.2 Share Notification Information

The PiCsMu share notification is used to notify a PiCsMu user that a file was shared and it is available to be fetched. Figure 3.4 shows the class diagram of a PiCsMu share notification and its attributes, also showing the reference to the File Information class.

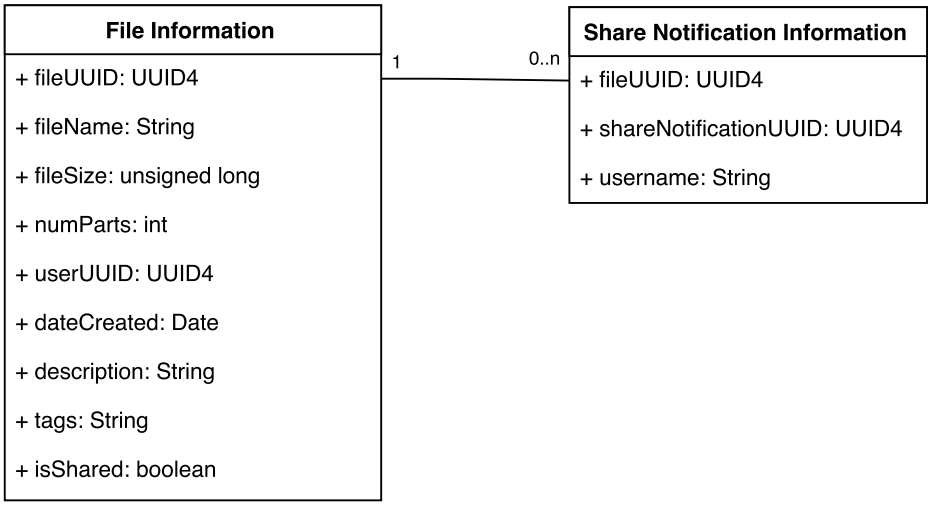

Figure 3.4: Class diagram representation of PiCsMu Share Notification Information

The attribute "fileUUID" corresponds to the unique identifier from the File Information class, and is used to find the particular shared PiCsMu index. The "username" attribute is used to inform which PiCsMu user shared such file. By using the "fileUUID" attribute on each PiCsMu share notification, the PiCsMu application is able to search the appropriate file in the PiCsMu P2P network and start the file download. The "username" attribute is used to fetch the corresponding public key from the PiCsMu IdP. Based on the public key, the PiCsMu application is able to verify the digital signature of the file and, therefore, also its integrity – i.e., verify whether a given PiCsMu user indeed shared the file or not.

3.4.3 Friendship Information

The PiCsMu social network uses a friendship relation that is unilaterally acknowledged, using publish/subscribe concept [34]. One PiCsMu user can "subscribe" to another PiCsMu user, thus establishing a unilateral friend-

ship. In the context of this thesis, the term "follow" is used to express the subscription between PiCsMu users. Thus, a PiCsMu user *X* is able to follow *n* PiCsMu users, but not necessarily each of the *n* follow PiCsMu user X back. In this case, *X* is called the *followee*, while each of the *n* are entitled a *follower*.

Figure 3.5 shows the information model to express friend relationships in the PiCsMu system. The attributes "follower" and "followee" are represented as an integer pointing to the PiCsMu user identifier ("userId" attribute in the *PiCsMu User Information* class), while the "dateCreated" attribute represents the date when the unilateral relationship was established. The *PiCsMu User Information* class represents PiCsMu users which are kept in the PiCsMu IdP.

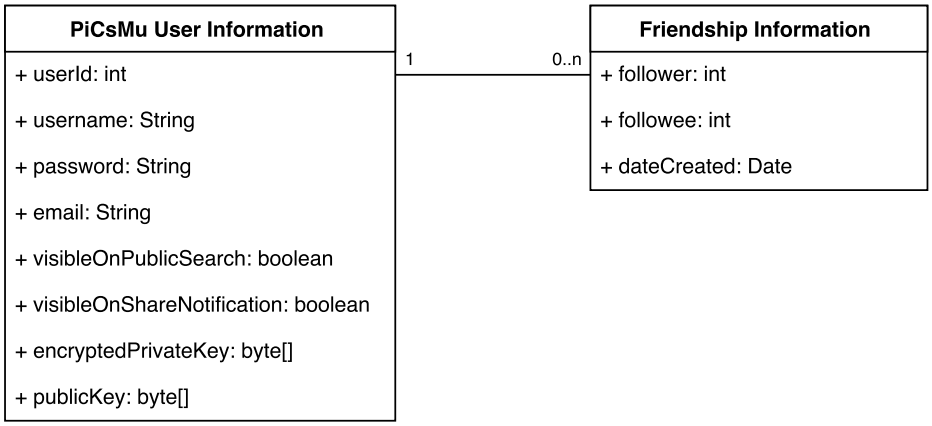

Figure 3.5: Class diagram representation of PiCsMu Friendship Information

The *PiCsMu Friendship Info* list is kept in the PiCsMu IdP since it could be manipulated by malicious peers if persisted in the PiCsMu P2P network. The list is then fetched from the PiCsMu IdP and transmitted to the PiCsMu application upon start. The list is updated on the PiCsMu IdP if a modification happens on the application side (e.g., follow/unfollow a PiCsMu user).

3.5 Storage Modes

The PiCsMu system presents 3 modes of operation: *private storage*, *private sharing*, and *public sharing*. Based on what the PiCsMu user selects for a file

upload, i.e., share a file or upload it only for private use, one of the modes of operation is invoked. While *private storage* mode keeps the PiCsMu index on a centralized entity (PiCsMu centralized index service), *private* and *public sharing* modes keep the index in a decentralized manner (PiCsMu P2P network). The *private sharing* mode also generates PiCsMu share notifications and keep them in the PiCsMu P2P network.

3.5.1 Private Storage

If a file should not be shared, i.e., kept private to the uploading PiCsMu user (private mode), the PiCsMu index information of the file is stored in the PiCsMu central index server. As shown in Figure 3.6, which illustrates the steps and interactions for the private storage mode, the PiCsMu central index server is responsible for authenticating and authorizing which PiCsMu user has access to which PiCsMu index entity. The authentication is performed by the PiCsMu IdP. The PiCsMu centralized index service receives an authentication request from the PiCsMu application containing the PiCsMu identity (username and password). Thus, the PiCsMu IdP can check the validity of the given PiCsMu identity.

Once the authentication is done, the PiCsMu centralized index service can authorize the particular PiCsMu user to have access to previously persisted private index entities. Therefore, the PiCsMu system guarantees that each PiCsMu user has the right to only delete or retrieve its own file index entities.

3.5.2 Private Sharing

Within the private sharing mode, as shown in Figure 3.7, a PiCsMu user (sender) can upload and share a file with one or more specific PiCsMu users (receivers). Moreover, receivers have the possibility to verify the sender within the PiCsMu IdP. The private sharing mode and the sender verification mechanism are described in 4 steps:

1. **Create a Sender's Digital Signature:** The sender creates a signature based on the index entity with the sender's private key.

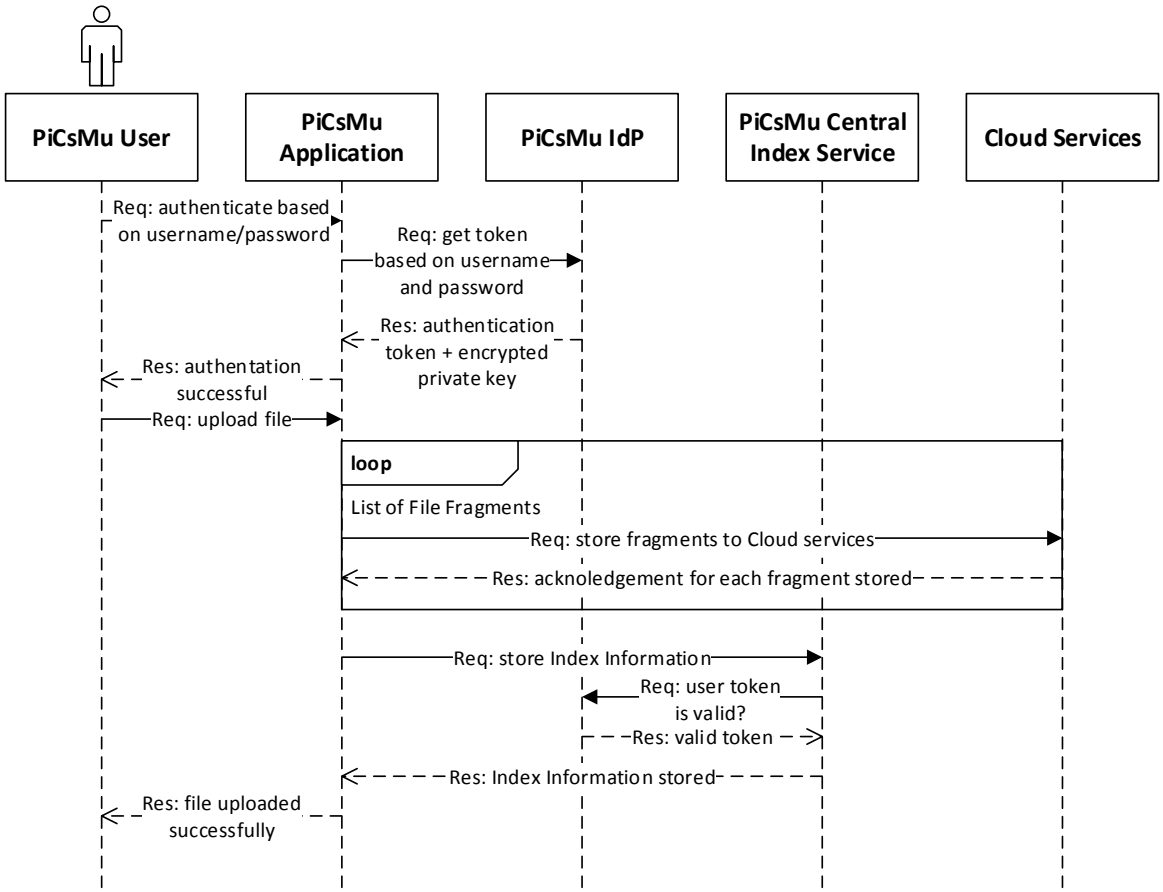

Figure 3.6: Sequence diagram of the private storage mode

2. **Encrypt the PiCsMu index entity:** The sender encrypts the top-level PiCsMu index entity using a randomly generated key (PBE algorithm).

3. **Encrypt the Random Generated Key:** The sender encrypts the randomly generated key (from step 2) with the public keys from the receivers.

4. **Concatenate Results:** All results of steps 1 to 3 are appended. The result of these steps applied for each PiCsMu index entities generates *PiCsMu encrypted index* entities. The PiCsMu encrypted index entity carries additional information and, therefore, ensures that the receiver obtains all cryptographic information necessary.

Every time a file is shared with another PiCsMu user, the sender is responsible for generating a PiCsMu share notification entity for each receiver. As the PiCsMu index, the PiCsMu share notification entity is also

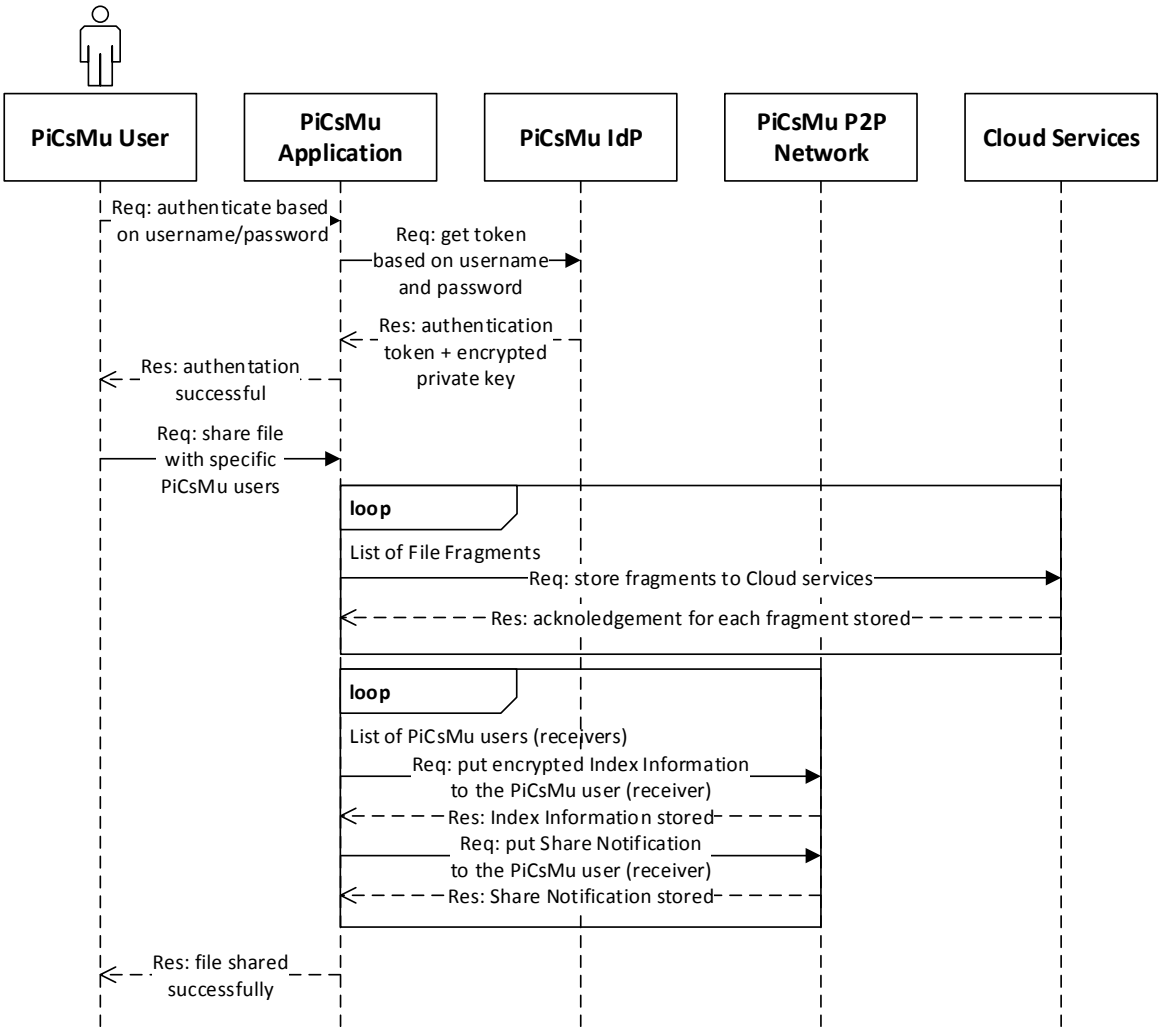

Figure 3.7: Sequence diagram of the private sharing mode

stored in the PiCsMu P2P network and it is used to notify receivers of shared files.

In order to avoid unwanted share notifications, a PiCsMu user only can receive files if the sender's PiCsMu identity exists in his/her PiCsMu Friendship Info list. If the PiCsMu system is only used for private storage or to share files with others (but not receive shared files), maintaining a PiCsMu Friendship Info list is optional.

3.5.3 Public Sharing

Within the public sharing mode, a PiCsMu user can upload and share a file with the entire PiCsMu system and, therefore, any PiCsMu user has access to it. As shown in Figure 3.8, which illustrates the steps of the public sharing

mode, it is not required to encrypt the PiCsMu index. Thus, each of the PiCsMu index entities are directly stored in the PiCsMu P2P network.

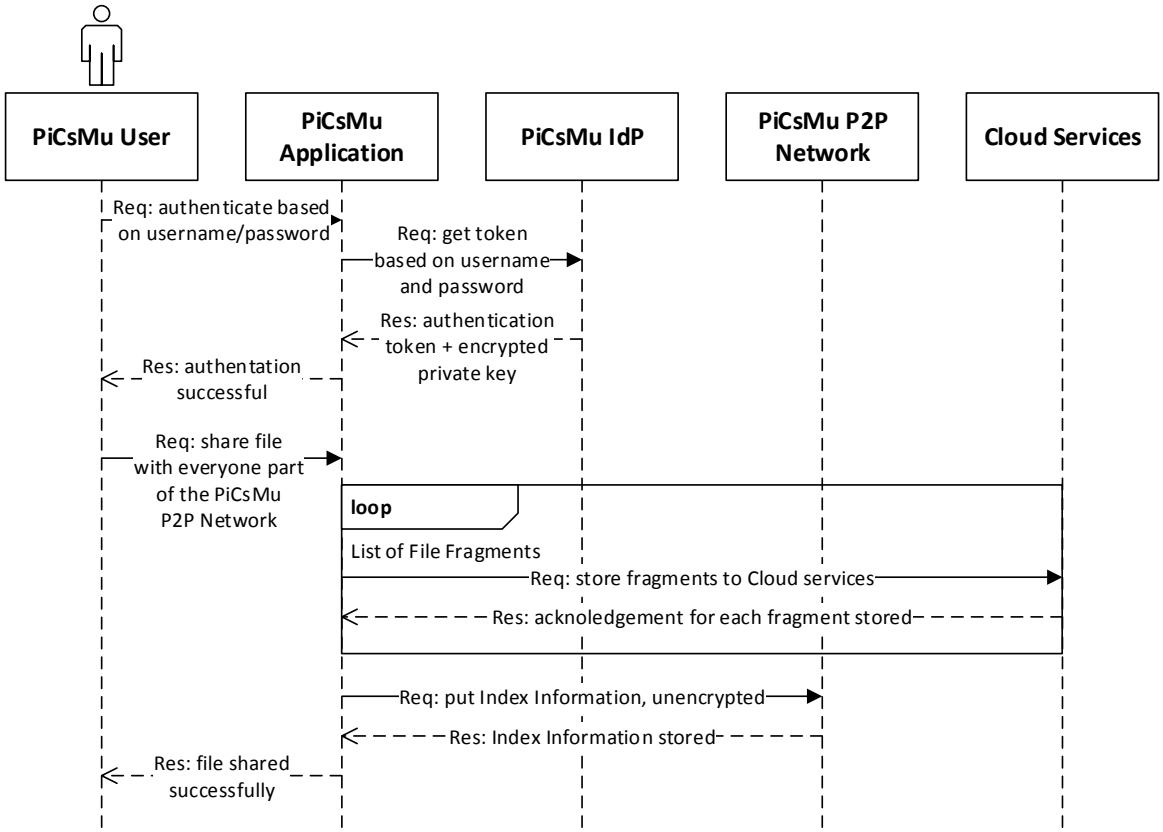

Figure 3.8: Sequence diagram of the public sharing mode

Instead of using PiCsMu share notifications, content can be searched through a content-based search query when stored through the public sharing. Content-based search used in PiCsMu allows for the lookup of files without knowing an exact keyword by using P2P FastSS [12] [11] [13], a similarity algorithm based on the edit distance metric (Levenshtein) [70]. Section 3.6.2 presents the design of the content-based search and required structures.

3.6 PiCsMu Peer-to-Peer Network

Considering the private and public sharing storage modes, both PiCsMu index and PiCsMu share notifications are stored in the PiCsMu P2P network. The PiCsMu P2P network uses a Distributed Hash Table (DHT)

provided by TomP2P [119] in order to keep these entities in a distributed manner.

Section 3.6.1 presents the design of a DHT structure scheme to store PiCsMu index and PiCsMu share notifications, while Section 3.6.2 presents how to enable a content-based search of files for the public sharing storage mode. The key point of this section is to present a DHT structure scheme that demands as low DHT calls as possible and, therefore, aiming to reach high performance. To achieve that, the design aims to make an optimal use of available functions in a DHT, more specifically being based on TomP2P features.

3.6.1 DHT Structure Scheme Design to Store the Information Model

Calls to a DHT are designed to follow a structure scheme to create and access entries according to application requirements. The design of a DHT structure scheme should avoid key collisions and unintended redundancy, since it can cause, e.g., malfunction of a certain feature and performance degradation. When calling the PUT operation using an already existent key, the available entry is overwritten (if it is not protected), or the call is ignored. Therefore, TomP2P allows the creation of additional identifiers to enable separate spaces for keys, which are called as *domains*. A domain is used together with a DHT key and builds a unique combination [6]. Implications include an equal DHT key that can be used multiple times in different domains, extending the key space (i.e., set of available keys) by a large factor – therefore, enabling a more efficient lookup.

The proposed approach introduces three domains [6]: (1) index domain, (2) search domain, and (3) notification domain. In the scope of (1), all values related to the PiCsMu index are stored. Domain (2) is used to store values that can be searched for (i.e., search terms that points to files in the index domain). As further explained in Sections 3.6.2, multiple keywords are stored for each file, demanding a large key space and justifying a separate domain. In addition, performance can be increased by limiting search space to a single domain. The notification domain (3) stores user

names and its associated share notifications. Each time a user privately shares a file with another user, a notification is placed in this domain of the receiver's user name. Files available to be downloaded can be seen by checking one's personal notification domain. Based on the same principle as (2), a separate domain is used to reduce the key space and increase lookup performance.

Table 3.1: Example of Put/Get operations using location and content keys

Use of Content Keys	
Put(location key, content key, value)	**Result**
Put(0x123, 1, object1)	-
Put(0x123, 2, object2)	-
Get(location key, content key)	**Result**
Get(0x123, 1)	object 1
Get(0x123)	object1, object2

Regular DHTs typically operate with one type of keys (i.e., location keys) to store and retrieve values. One key is associated with one value. However, there is often a need to store multiple values under the same key. Domains are able to solve this issue, and the specification of TomP2P allows for the differentiation between *location keys* and *content keys* to design a DHT structure scheme. By storing values under the same location key, but with different content keys, it is possible to retrieve all values at once using one GET call. An example is given in Table 3.1. Two objects (object1, object2) are stored using the same location key (0x123) but with different content keys (1 and 2). With GET, either one specific value or all stored values can be retrieved [6].

Based on the domain and content keys concept, the DHT structure scheme to store the PiCsMu index is shown in Table 3.2 [6]. Location and content keys are hashes of certain values. Hash functions (e.g., SHA-1 [91], SHA-2 [91], MD5 [105]) create fixed length keys (i.e., hashes) from variable length data. A hash function will always create the same key for

Table 3.2: DHT structure scheme used to store the PiCsMu index, search keywords, and share notifications

Distributed Hash Table

Location Key	Content Key	Value	Domain
hash(FileUUID)	*hash*(File)	File	Index
hash(FileUUID+UserID)	*hash*(*enc.*(File))	*enc.*(File)	Index
hash(FileUUID)	*hash*(FilePartID)	File Part	Index
hash(FileUUID+UserID)	*hash*(FilePartID)	*enc.*(FilePart)	Index
hash(CredentialUUID)	*hash*(Credential)	Credential	Index
hash(CredentialUUID+UserID)	*hash*(*enc.*(Credential))	*enc.*(Credential)	Index
hash(UserID)	*hash*(CredentialUUID)	*enc.*(Credential)	Index
hash(UserName)	*hash*(FileUUID)	Notification	Notification
hash(Keyword)	*hash*(FileUUID)	FileUUID	Search

equal input. The hashes are important for the DHT's underlying routing algorithm. The abbreviation "*enc.*" stands for *encryption*, while "Notification", "Credential", and "File" appearing in the "Content Key" and "Value" columns represents the Share Notification Information, Credential Information, and File Information entities, respectively.

In private sharing, all objects put to the DHT are encrypted using the public key of the receiver. Table 3.2 also shows the benefits of content keys. Since file parts are stored under their equal file UUID, all parts can be retrieved using one `GET(fileUUID)`, instead of having one DHT call for each file part. Reducing the number of calls on the DHT increases performance and is less susceptible to failures.

3.6.2 Content-Based Search

One fundamental feature of file sharing systems is the possibility to search for available files. DHT-based systems lack the support of a user friendly content-based search and can only do exact match search by default. Content-based search allows the lookup of files without knowing an exact keyword. Results similar to the specified search term are presented. The reason why DHTs are intrinsically unable to perform such a search is the use of hash functions. Calculating a hash on similar keys results in com-

pletely different hash values. As an example, if a file is stored using its file name as key, it can only be found by a DHT search if the exact same name is used again. If only one letter in the name is different, the hash function will create a totally different key and the DHT can not retrieve the file. This is a non-desired behavior when trying to search for files without knowing their exact keywords. Therefore, some P2P systems (e.g., BitTorrent, Freenet) do not support a direct key search but use other structures, like, e.g., a Web service providing search based on the content found in the P2P network.

PiCsMu system is designed to support a content-based search by using the P2P Fast Similarity Search (P2PFastSS) [12][11][13], which is a similarity algorithm based on the edit distance metric. The edit distance, generally referred to as Levenshtein distance, is a string metric to calculate the minimum number of operations required to transform one word into another, using the operations: deletion, insertion, and replacement [70]. For example, the edit distance between *house* and *mouse* is 1 because the only operation needed to transform *house* into *mouse* is to replace *h* with *m* [11]. Based on such transformations, the algorithm defines a deletion neighborhood to search for similar words. The deletion neighborhood of a word is the set of all strings (i.e., neighbors) created by deleting each character of the word once (edit distance=1). Table 3.3 shows an example by generating the deletion neighborhood of the word *fast*.

Table 3.3: Deletion neighborhood of the word *fast* with edit distance=1

fast	
Deleted Character	Neighbors
f	ast
a	fst
s	fat
t	fas

Enabling a content-based file search requires three steps to take place for each uploaded file [6]: (1) generation of keywords, (2) generation of the deletion neighborhood for each keyword, and (3) indexing all created neighbors. Step (1) defines all keywords related to a file that a user is able

to search. When storing a file using PiCsMu, a user can define certain attributes which the content-based search will use to locate the file. The composition of a file is described in Section 3.4.1 and the relevant attributes to enable the content-based file search is summarized in Table 3.4. Each attribute, except the file description, is included to the set of keywords as it is. Therefore, a user will be able to search for a file name or tag directly and receive a list of matching files. Since the file description is a text in natural language, it contains words that are unsuited to be keywords in a search, e.g., prepositions, conjunctions and pronouns. Such words, called *stop words*, are widely used in the English language and can not identify a single file. Hence, the description needs to be filtered to identify a number of keywords that stand for the complete text. Therefore, a simple tag cloud algorithm is used: it removes the stop words from the description, counts the frequency of occurrence of remaining words, ranks them in descending order, and adds the top results to the set of keywords [6].

Table 3.4: User-defined attributes of a file

Concept of a file	
Attribute	Description
File name	Simple file name
File description	A short text about the file
Tags	List of words that categorize the file
User name	PiCsMu user name of the uploader

Upon generating all keywords, step (2) creates the deletion neighborhood for each keyword as shown in Table 3.3. Step (3) is concerned to index all search keywords and their generated neighbors, by putting them in the DHT. In the following explanation, only the term *keywords* is used to describe the combination of search keywords and their neighbors. As explained in Section 3.6.1 and illustrated in Table 3.2, keywords are assigned to an individual domain in the DHT. They are stored using the keyword itself as location key and the UUID of the file they belong to as content key. Therefore, different files but with similar keywords have their UUIDs stored under the same location keys.

After completion of all three steps, the file can be found by the content-based search. Search results are based on the joint possession of neighbors. Table 3.5 shows a neighbor comparison of the words *east* and *west*. By having at least one neighbor in common (here: *est*), the two words are considered to be similar. If a user searches for a word, step (2) of the above described process is applied again. For each generated neighbor, a DHT GET operation is done, using the neighbor as a key. Thus, the user gets all file UUIDs, with at least one common neighbor of the word and the generated keywords of a file. With a list of file UUIDs (i.e., the search result), selected files can be downloaded if desired.

Table 3.5: Comparison of the deletion neighborhood of two words with edit distance=1

Neighbor Comparison		
Operation	east	west
Deletion of character 1	ast	est
Deletion of character 2	est	wst
Deletion of character 3	eat	wet
Deletion of character 4	eas	wes

3.7 Data Encoders

In order to build a system that is able to *aggregate* several different data-specific Cloud storage services besides generic ones, it is required a mechanism to turn these data-specific Cloud storage services into a pool of homogeneous entities – thus, being possible to request to store any kind of data, independently of specific characteristics from Cloud services. The challenge lies on developing a mechanism that does not require any change on the Cloud service side, such as, e.g., APIs, service's implementation, or business strategies.

PiCsMu uses *data encoders* to turn a pool of data-specific Cloud storage services into homogeneous entities for data persistence. Instead of requiring any change from Cloud service's side, the entity demanding to store data

– in this case, PiCsMu – adapts the data to adequate itself to rules defined by data-specific Cloud storage services. Thus, PiCsMu transforms data into a representation that fulfils the requirements of Cloud services. For example, if data must be represented as a picture, thus, *data encoders* inject such data into a JPG image using different possible techniques. The data encoding mechanism brings advantages in terms of flexibility, since it can be adapted by any Cloud services with different kind of restrictions. However, it brings data overhead depending on which data encoding mechanism is employed.

Data encoders' design details and implementation are presented on Chapter 4, as well as a methodology on how to analyze data validation of Cloud services based on tests performed applying these data encoders.

3.8 Data Reliability

As in any storage media, data can be lost when stored in Cloud services [16][100][10]. Thus, to tackle data reliability, the PiCsMu system uses Forward Error Correction (FEC) to be able to recover data portions that were lost. FEC introduces redundancy to transmitted information using a predefined algorithm. The redundant bytes are produced by complex functions of the original information bytes. Such bytes are sent multiple times, since "errors" or data loss may happen in any of the uploaded fragments. In contrast to the $n+1$-parity technique where redundancy is reached through XOR operations [102], FEC algorithms can recover from multiple simultaneous failures (i.e., lost fragments). Thus, FEC is particularly suitable to be employed in the PiCsMu system since it stores data in multiple independent Cloud services which can present simultaneous failures, such as unavailability, accounts being deactivated, and legal issues.

The PiCsMu system uses Reed-Solomon code as the underlying mechanism for the PiCsMu Forward Error Correction, or simply called as PiCsMu FEC. The main concepts related to Reed-Solomon are shown in Appendix A.1, as well as the terminology of symbols used throughout this section.

One important aspect is that the PiCsMu system is primarily concerned with erasures and not with errors. During the file download process, the PiCsMu application is able to detect whether a data fragment is not avail-

able for download or the fragment is somehow lost – e.g., when accounts are shutdown without user's consent, or when data is not available due to Cloud service's infrastructure problems. Such detection is beneficial since Reed-Solomon is able to recover twice as much data symbols when identifying which fragment was lost.

The PiCsMu FEC, is composed of the following steps:

1. **FEC Encoding**: Based on the fragmentation step (cf. Section 3.3), generate n fragments given that p fragments can be lost for a successful file reconstruction. The decision of choosing the p value, and consequently the n value, depends on factors that are not discussed in this thesis, but it is presented as a configuration parameter. For example, the p value can be chosen based on how many different Cloud service's credentials the PiCsMu user specified, since several different Cloud services without any commercial relation tend to be more independent related to IT infrastructure and less prone to simultaneous unavailability issues. Thus, depending on user's redundancy requirements the p value should be higher or lower.

2. **File Downloading**: Observe if any of the n fragments generated in the FEC Encoding step cannot be downloaded. Based on the PiCsMu index (cf. Section 3.4.1), more specifically looking to the number of file parts, it is possible to distinguish which file fragment is missing.

3. **FEC Decoding**: Based on x downloaded fragments and the missing file fragments, perform the decoding and reconstruct the original data. If the downloaded fragments are lower than $n - k$, it is impossible to reconstruct the file and the PiCsMu application should retry the File Downloading process once again.

As an illustrative example of the PiCsMu FEC process, let us assume a PiCsMu User that provided 4 different Cloud services credentials and specifies a 10 MByte file to upload. If the fragmentation step splits the file in 10 fragments of 1 MByte, and the p is set to 4 – same number of Cloud services'

credentials – the n value is 14. Thus, the Reed-Solomon code is defined as a RS(14,10) and is able to correct 4 simultaneous erasures.

3.9 Use Case

To demonstrate the file upload and download processes, a use case is provided for the private file sharing [73]. The private file sharing is illustrated in a use case since (a) it uses all the functionalities presented in this chapter, and (b) it involves more than one PiCsMu user to upload/download files. Figure 3.9 illustrates Alice sharing a file specifically with Bob. "PM-ID" represents the unique identifier of a PiCsMu identity, having "PM-ID A" as Alice's and "PM-ID B" as Bob's identifiers. It is assumed that Alice and Bob created their PiCsMu identity using the PiCsMu application, in the same PiCsMu IdP, at an earlier stage.

Both Bob and Alice start their PiCsMu application, and it automatically fetches her and his public and encrypted private key, respectively, from the PiCsMu IdP (step 1). The encrypted private key is decrypted with that user password, which was chosen when the PiCsMu identity was created to obtain the private key. In parallel, but still in step 1, their respective PiCsMu applications contact the PiCsMu P2P bootstrapping servers to connect to the PiCsMu P2P network, also fetching their friends information (PiCsMu Friendship Info) from the PiCsMu IdP.

In step 2 Alice provides to her PiCsMu application the file to be shared with Bob and one or more Cloud services' credentials. Once the PiCsMu application receives the file and all Cloud services credentials are in place (specified by Alice), it starts the file upload process as described in Section 3.3, including the PiCsMu FEC encoding. During the file upload process her PiCsMu application recognizes that the file is shared (with Bob) and within step 3 Alice's PiCsMu application requests Bob's public key from the PiCsMu IdP, including Bob's PM-ID.

At step 4 Alice's PiCsMu application uploads the encoded file parts to those Cloud services Alice provided credentials for. Once the uploading of all file parts is successfully completed, Alice's PiCsMu application encrypts each PiCsMu index entities with a randomly generated key, which is also

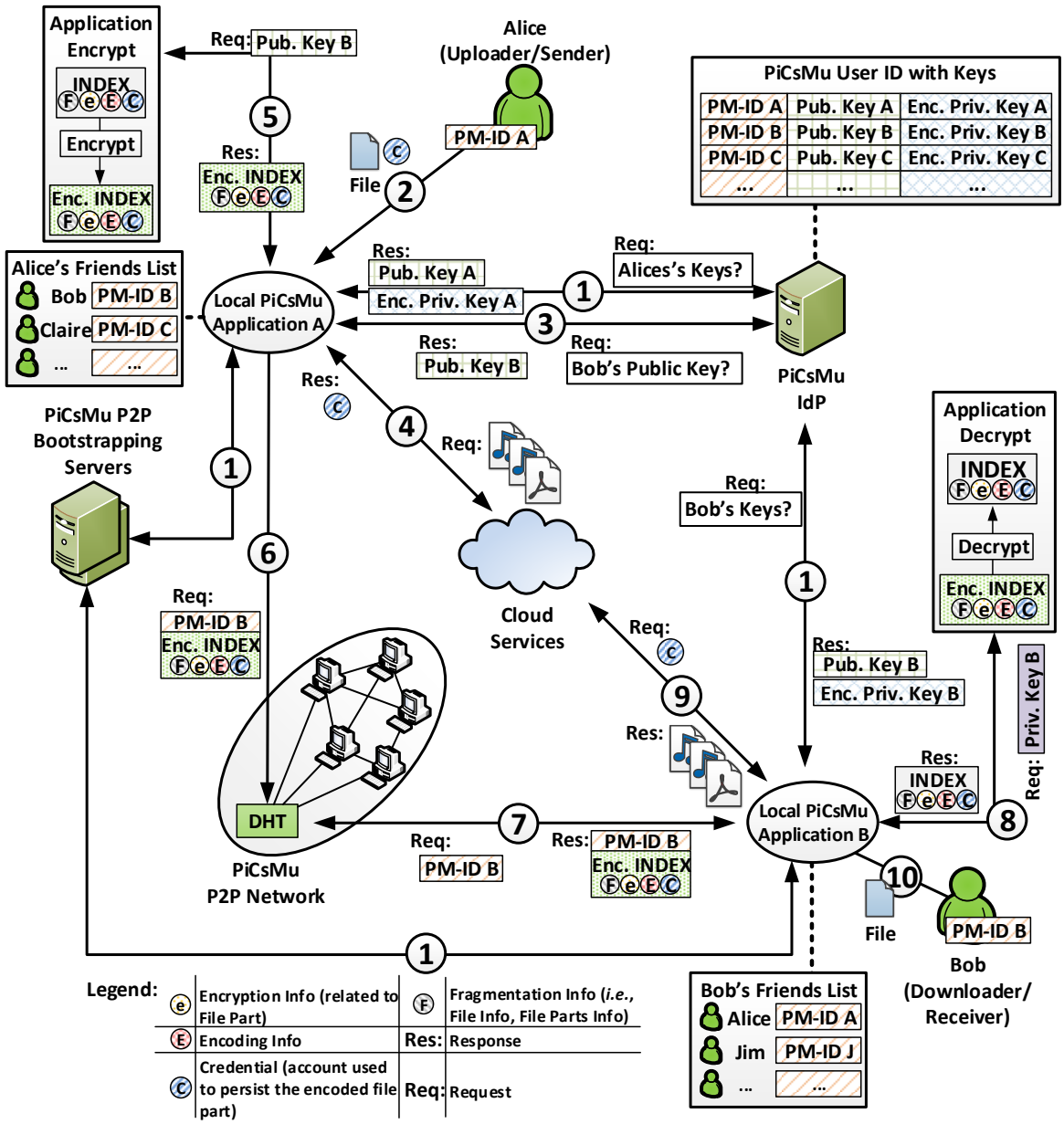

Figure 3.9: Private sharing steps

encrypted with Bob's public key (step 5). Moreover, Alice's digital signature is included with each of the PiCsMu encrypted index entities.

In step 6 Alice's PiCsMu application persists each of the PiCsMu encrypted index entities to the DHT. Therefore, Alice creates a PiCsMu share notification entity to make Bob aware of the newly shared file.

Bob's PiCsMu application consults the DHT for PiCsMu share notification entities that was sent by his friends. Once Bob recognizes a new notification, here from Alice, Bob's PiCsMu application uses the file UUID from the PiCsMu share notification to start a sequence of DHT calls that

are necessary to retrieve all PiCsMu encrypted index entities that forms the PiCsMu index (step 7).

Holding all PiCsMu encrypted index entities of the shared file, Bob's PiCsMu application verifies in step 8 the digital signature from step 5 against Alice's public key provided by the PiCsMu IdP. In parallel, Bob's PiCsMu application is able to decrypt the PiCsMu encrypted index entities by using Bob's private key.

Finally, within step 9 all Cloud services' credentials, part of the index, are used to locate all encoded file parts in the Cloud services storage. Upon retrieving all encoded file parts, Bob's PiCsMu application follows the file download process, including the PiCsMu FEC decoding, and reconstructs the original file shared by Alice.

3.10 Prototype Implementation

A Java-based prototypical implementation of the PiCsMu System was developed [73][77] to verify its feasibility, scalability, and overhead. The PiCsMu application implements the following modules using well defined interfaces for reusability: *Schedulers*, *DataEncoders*, *FECStrategies*, *EncryptionStrategies*, *IndexPersistenceStrategies*, and *CloudServices*. The security modules implement *PBE-AESCryptography*, which is used to encrypt larger amounts of data, such as PiCsMu index entities and data fragments, while the *RSACryptography* just encrypts smaller amounts of data, such as a randomly generated key. The *PBE-AESCryptography* handles a key length up to 128 bit. The Forward Error Correction module has one implementation, the *ReedSolomonFEC* class, being extensible to include more FEC strategies. The *ReedSolomonFEC* class has methods to encode (`encode()`) and to decode data (`decode()`), being possible to set the appropriate Reed Solomon code configuration, i.e., RS(n,k), in the class constructor.

CentralizedIndex and *DistributedIndex* classes, which are part of the *IndexPersistenceStrategies* module, implement the persistence of PiCsMu index entities in the PiCsMu central index server and in the PiCsMu P2P network, respectively. Both implement methods to persist PiCsMu index entities, such as `putFileInfo()`, `putFilePartInfo()`,

`putCredentialInfo()`, as well as methods to retrieve the PiCsMu index entities, such as `getFileInfo()`, `getFilePartInfo()`, and `getCredentialInfo()`.

The *CentralizedIndex* implementation expects as input the PM-ID and PiCsMu identity password (encrypted using SSL) for each *get* or *put* called methods, since the PiCsMu central index server is implemented with stateless calls (i.e., without the notion of a session). Therefore, *get* or *put* methods can be invoked any time in a Remote Procedure Call (RPC) fashion encoded in JSON (JavaScript Object Notation). The communication between the *CentralizedIndex* implementation and the PiCsMu central index server is done following the JSON-RPC standard [61]. The PiCsMu central index server and the PiCsMu IdP are implemented as standalone Java programs, receiving and answering JSON-RPC method calls. In the current prototype implementation, both PiCsMu central index server and PiCsMu IdP run a MySQL database to persist all PiCsMu index entities and PiCsMu identities.

The *DistributedIndex* implementation follows the DHT structure presented in Section 3.6.1, using the TomP2P library [119]. It allows the PiCsMu application to *put* or *get* PiCsMu index entities, search keywords, and PiCsMu share notification entities to/from the DHT. The PiCsMu bootstrapping servers are also implemented as standalone Java programs to receive bootstrapping requests from the PiCsMu application.

The PiCsMu social authentication server (PiCsMu SAS) is implemented as a standalone Java program with a REST API interface to request authentication and authorization (e.g., OAuth) to OSNs. The REST API implements the asynchronous method `requestAuth(parameters)`, which start the OAuth process using the OSN credentials specified in the parameters. The OAuth protocol flow [95] requires a callback URL that OSNs can call to provide the access token for such authentication and authorization request. Thus, the PiCsMu SAS's IP address is used and it should be accessible from OSN's servers.

3.10.1 Cloud Services

Each of the CSs supported by PiCsMu were implemented having 4 basic methods as the interface to the PiCsMu application: (a) `put()`, (b) `get()`, (c) `getMaxInputSize()`, and (d) `getAllowedFileFormats()`. In (a) and (b), the Cloud service implementation should be able to store and retrieve files, respectively, having a Cloud service credential as parameter. While in (a) the method returns a unique identifier to locate the file within the scope of the Cloud service account associated with the specified Cloud service credential, in (b) the method expects this unique identified as parameter to retrieve the correct file. In (c) the implementation specifies what is the maximum file size that this Cloud service can handle for each upload, and in (d) the Cloud service implementation returns the allowed file formats to be uploaded. Based on the allowed file formats the PiCsMu application can check what are the possible data encoders to be used with each Cloud service. The following Cloud services are supported by the PiCsMu system prototype:

CSG Service: The CSG Service is an internal storage service in the CSG group at the University of Zurich. The *CSGService* implementation uploads a file, without file format restrictions, up to 20 MByte, and it returns a URL with a randomly generated identifier to fetch the file again, without the need of a Cloud service credential. Credentials are necessary to upload, but not to download files. The CSG Service is classified as a generic Cloud storage service.

Google GMail: Google GMail is an email service which allows to store data within the email body and email attachments. *GMailService* implementation uses the structure of an email to store data, thus being classified as a data-specific Cloud storage service. It creates an email to be sent to the same GMail account associated with the specified Cloud service credential, storing the encoded file as an attachment. The attachment maximum file size is 25 MByte, and the email body is left blank. In order to identify in which email the encoded file is located, the *GMailService* implementation

generates a random UUID in the email subject. The *GMailService* implementation accepts any format.

Facebook Update Status: Facebook is an OSN providing the status update as a feature. Facebook users can write a text input which has a maximum input size of 63,206 characters by today. The *FacebookStatusUpdateService* implementation uses the status update field to store data, thus, being classified as a data-specific Cloud storage service. Every time that a status update is created, the Cloud service generates a unique identifier associated with the status updated. Therefore, data represented as text is the input, while a URL generated by the Cloud service that locates the status updated is the output. The Cloud service credential used to update the status is also necessary to retrieve the status update. *FacebookStatusUpdate* Service only accepts text as input.

Google Picasa: Google Picasa is an image organizer and viewer. Google Picasa accepts the upload of images with sizes up to 20 MByte, despite of the image resolution, thus, being classified as data-specific Cloud storage service. The *PicasaService* implementation takes an image file as input, generates a random UUID which is associated with the Picasa image name attribute, and returns such a UUID as an output. The UUID is valid in the scope of the Cloud service credential given to upload the image, therefore, the used Cloud service credential to upload is also necessary to download the file. Although the Google Picasa service accepts a wide range of image file formats, the *PicasaService* implementation prototype only accepts PNG and JPG formats.

ImageShack: ImageShack is an image organizer and viewer. It accepts the upload of images with sizes up to 5 MByte, despite of the image resolution, thus, being classified as data-specific Cloud storage service. The *ImageShackService* implementation takes an image file as input and returns a URL with a randomly generated identifier to fetch the file again, without the need of a Cloud service credential. A Cloud service credential is necessary to upload, but not to download the file. The *ImageShackService* imple-

mentation prototype accepts PNG and JPG files, even though the Cloud service accepts a wider variety of image formats.

SoundCloud: SoundCloud is a service platform which allows users to upload, organize, and listen to audio. It accepts the upload of audio files, without a maximum file size, but with a free account limiting to 60 minutes of audio. SoundCloud is classified as being a data-specific Cloud storage service. The *SoundCloudService* implementation takes an audio file as an input and returns a URL with a randomly generated identifier to fetch the file again. The Cloud service credential used to upload is also necessary to download the file. The *SoundCloudService* implementation prototype only accepts MP3 files, even though the Cloud service accepts generally a wider variety of audio formats.

3.10.2 Scheduler Strategy

The *PiCsMu scheduler* component chooses how to fragment files, which encryption to use, which data encoder to select, and finally to which Cloud service to upload. Different *Scheduler* implementations may prioritize different aspects, e.g., to lower number of file parts, to lower number of used Cloud services, or to higher the number of Cloud services. The *DefaultScheduler* implementation uses a random function to determine Cloud services and its credentials, data encoders, and encryption strategies for each file part.

The fragmentation strategy followed by the *DefaultScheduler* is to try to put as much data within one file part, considering the chosen Cloud service and data encoder. The steps performed by the *DefaultScheduler* implementation are shown in Figure 3.10. First, step 1 obtains the file size from the file to be uploaded. Step 2 retrieves the list of Cloud services' credentials provided by the PiCsMu user, also creating a Fragmentation Plan list. Each element of this list is a Fragmentation Plan object, which represents information about how a particular fragment should be manipulated by PiCsMu, e.g., the file offset where the fragment data start, the fragment size, which data encoder that will be considered for this particular fragment, which file base the fragment data will be injected, which Cloud service it

will be stored, and which Cloud service's credential it will be used. While the entire file is not considered for the fragmentation planning (step 3), steps 4 to 13 are executed.

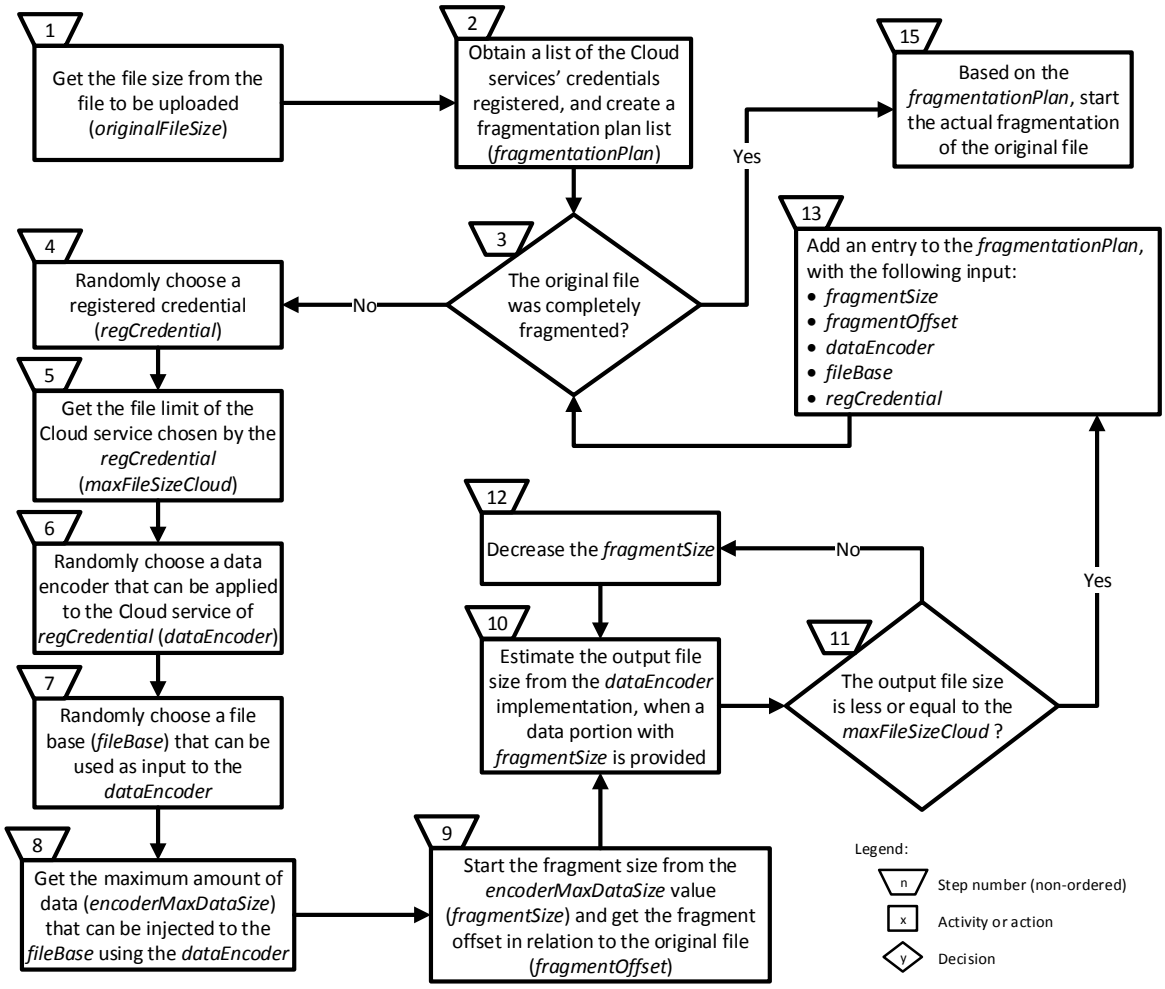

Figure 3.10: Flowchart of the PiCsMu scheduler implementation

The steps 4 to 8 randomly choose credentials, data encoders, and file bases, as well as obtaining what is the maximum amount of data that can be injected in a file base using the chosen data encoder. Based on the information from previous steps, step 9 sets a fragment size. Such fragment size is used to get an estimation of what size the file output will have if the chosen data encoder and file base are used (step 10). If the estimation is beyond what the randomly chosen Cloud service can accept as a file size, then the fragment size should be decreased until the file size restriction is satisfied (loop in steps 10, 11, and 12). The current implementation decreases the fragment size in 10 Byte. When the restriction is satisfied, a Fragment Plan

object is created containing all the information for the specific fragment, and it is included in the Fragment Plan list (step 13). All these steps are repeated until there no data portion left to be considered by a Fragmentation Plan entry. Since each of the Fragmentation Plan entries that constitutes the Fragmentation Plan list are structures calculated by the *DefaultScheduler*, the actual fragmentation should be executed. Thus, finally, step 15 starts the upload process of the original file (cf. Section 3.3) providing the Fragmentation Plan list as input.

4

Data Validation in Cloud Services

The understanding on how Cloud services accept data to be stored in their servers is an important aspect to the PiCsMu system. It is necessary to be aware if data fragments can be stored in specific Cloud services and in which file format representation they must be encoded in order to be compliant to data validation processes of Cloud services.

This chapter presents an investigation of Cloud services' data validation processes, also checking if specific data encoders can be used in the PiCsMu system. First, it discusses why the understanding of data validation processes in Cloud services is important. Secondly, the method definition used to perform such analysis is presented. In the end, the implementation of several data encoders are shown, also presenting the proof-of-concept architecture and implementation used to perform the tests for the analysis of data validation of Cloud services.

The PiCsMu system uses the implementation of the data encoders presented in this chapter. Thus, this chapter also evaluates on which extent data encoders are suitable to be used in the PiCsMu system.

4.1 Importance on Understanding Data Validation of Cloud Services

The definitions of *generic* and *data-specific* Cloud storage services are mutually exclusive and the opposite of each other considering the following affirmations: if a pool of data-specific Cloud storage services become *homogeneous* in how they store data (e.g., same file formats or file's attributes), then such pool of Cloud storage resources can be called generic Cloud storage services; and, if a pool of generic Cloud storage services become *heterogeneous* (i.e., different file formats or file's attributes), then such pool of Cloud storage resources can be called data-specific Cloud storage services.

Data-specific Cloud services, usually represented as SaaS applications, verify whether data can be accepted or not by employing a set of rules following a process. This process, which is called *data validation*, should be considered by developers of data encoders. Depending on how the data validation is performed, data encoders should have characteristics to be able to generate files compatible to its data validation rules. Therefore, data validation of Cloud services should be understood – technically and impact-wise – in order to evaluate if the use of data encoders is feasible to homogenize a pool of data-specific Cloud services and, consequently, aggregate them as one single storage entity.

Understanding data validation processes is not only important for Cloud storage services overlays such as PiCsMu, but also for Cloud services themselves. A deep investigation on how data validation rules behave with several data formats and different data encoding mechanisms give Cloud services the opportunity to enhance their data validation processes to minimize negative impacts.

4.2 Methodology

The methodology employed to analyze Cloud providers' data validation is based on tests carried out on selected public Cloud services – the selection criteria is described in Section 6.1.1. The tests are classified as exploratory testing, which is defined by [8] as "simultaneous learning, test design, and

test execution". This means that the tester actively controls the design of test cases and its execution, using the gained information to design new and better tests. No research ever directed test cases to Cloud services' data validation process before [81]. Therefore, the selection of Cloud services, the designed test cases, and the implemented data encoders do not follow any previous results.

4.2.1 Test Objectives

The tests presented in this section evaluate if the data validation process can be bypassed using specific techniques in order to inject arbitrary data into Cloud services' servers. The tests objectives are directed to two impact dimensions:

1. **Security:** Evaluate if a file can carry data (data inside data) which the Cloud service is not aware of, since the data validation process and content filtering system cannot detect it. E.g., encode a PDF inside an image file, which is a file format that is accepted by the Cloud service. Thus, the PDF content cannot be seen by the service itself, which may contain, non-authorized information.

2. **Accounting and Charging:** Evaluate if a file encoded in different ways, and successfully uploaded to a Cloud service, can impact on how the Cloud provider accounts consumed resources and charges users. In this case, the goal is to check if a file can negatively or positively influence the amount of resources the user consumed – also observing if the Cloud provider is aware of such consumption. E.g., encode a PDF inside an MP3 file, making the MP3 file very big but just with 10 seconds of audio. In this example, the user would benefit of storing an amount of data that is not compatible to 10 seconds of audio (even though the MP3 is encoded using the best quality) and therefore taking advantage of storing data in Cloud services' servers. Considering that the user can store 120 minutes of audio in a free account, then it could be possible to deceive the accounting/charging

system in order to host many audio files of 10 seconds – but storing a huge quantity of data.

4.2.2 Test Method

The employed test method emulates users pushing and retrieving data of a SaaS application. After the data is pushed to the application, this method assumes that data flows directly through the data validation process. If the data gets validated by validation rules, it is accepted, persisted, and the SaaS makes it available for retrieval [81]. The reason to also retrieve the content is due to the necessity of checking if the data constructed (through the means of encoders, where it will be discussed in Section 4.3) was maintained or modified.

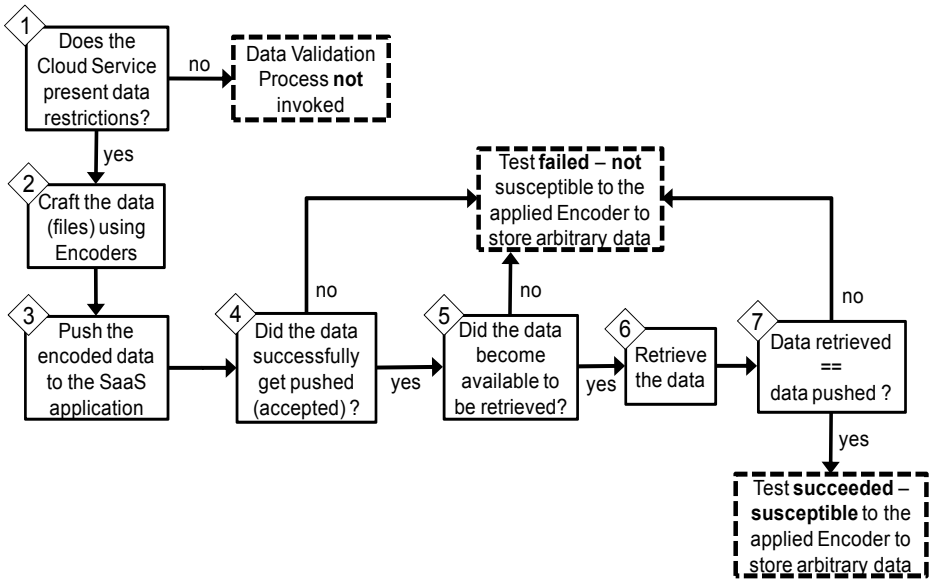

Figure 4.1: Test methodology steps

The methodology to test data validation in Cloud services is based on the following 7 steps (cf. Figure 4.1):

Step 1. For a particular Cloud Service, check what are the restrictions to push data to the application. If no restrictions are present, the data validation process will not be invoked. Therefore, the tests should consider encoders which may explore accounting and/or charging impacts. E.g., Dropbox allows its users to push anything to their account, limiting only the total space used.

Step 2. Based on the restrictions, construct the data to be pushed in a particular manner using encoders (cf. Section 4.3). The data should be encoded in order to answer the research questions from Section 1.2 and based on the data restrictions observed in Step 1. This process is key for the tests, since encoders can explore detailed factors in the data validation process.

Step 3. Push the data to the SaaS application. The data should be pushed following the Cloud service's API.

Step 4. Check if the data was successfully pushed (accepted) and became available to be retrieved. If the data was not accepted, the data validation process possibly detected a rule violation. Therefore, the Cloud service is not susceptible to the employed encoder constructing the data (unsuccessful test result). Note that in blackbox testing it is impossible to observe the exact reasons for the denial. If the data was accepted, the data validation process is susceptible to the employed data encoder.

Step 5. In case of a successful push, retrieve the data that was pushed in the previous step.

Step 6. Based on the retrieved data: if the data is modified from its original encoding (prior to the step 3) the data validation process has a mechanism to reorganize the data preventing that malicious users inject data to retrieve it in the original format. Therefore, the data validation process is not susceptible to the employed encoder – being impossible to re-construct the original data again. If the retrieved data remained the same compared to the step before encoding, the test case is considered successful. Moreover, if the Cloud service wrongfully accounted consumed resources – according to what the Cloud service accounts and charges for – its data validator led the accounting system to wrong values.

Steps 1, 3 and 5 are service-specific and therefore require a particular effort for each tested Cloud service, i.e., manually analyze Terms of Services contracts and upload/download data using an specific API.

4.3 Data Encoders and Proof-of-Concept Implementation

Data Encoders (or *encoders*) are software implementations that construct files (data) in a particular manner to test Cloud services' data validation. Encoders implemented in the scope of this thesis are divided into three groups: Steganography-related (Section 4.3.1), FileFormatHeader-related (Section 4.3.2), and Appending-related (Section 4.3.3). The encoder groups are defined by this thesis and they are used to both sort similar techniques, as well as to express the level of difficulty to encode and detect files that were encoded by such methods. More specifically, the encoder groups represent extremes of a trade-off between computational effort to encode and detect [81]: encoders part of the Steganography-related group are more difficult to be encoded and detected than encoders of the FileFormatHeader- or Appending-related group. Moreover, the encoder groups are used to express a generalization of data encoders to other file formats, but using the same technique. For example, it is possible to use FileFormatHeader-related encoders with any file format that has header fields.

The encoders designed and implemented in this thesis are not an extensive list of what is possible to explore in Cloud services' data validation tests. It just serves as a demonstrative sample of what could be developed from three distinct encoding groups. Each of the implemented encoders can present parameters, e.g., to vary the amount of data, or to inject data in different headers.

4.3.1 Steganography-related Encoders

Steganography-related encoders use the steganography technique, where messages (or, in this case, data) are hidden in a way that intends to turn them imperceptible apart from the sender and recipient. Steganography is a broad topic of research [59][41] and depending on how it is applied it can bring a high complexity to be detected. In this thesis, the term steganography is applied when data is injected in file's content data (e.g., slightly

modifying RGB bits in images, and WAV bits in audio files) that is still not perceivable by humans [64].

The following Steganography-related encoders were developed [81]:

JPG & PNG Steganography Encoder (*JPG-PNG-Stega*): The purpose of this encoder is to inject data (hiding it) in JPG and PNG files. It takes a *File Format Sample* (JPG or PNG) and, for each pixel, it injects 3 bits of data in the LSBs (Least Significant Bit). There are more sophisticated image steganography methods already widely discussed [64]. However, the use of this encoder shows that even basic methods of JPG and PNG steganography are possible in SaaS applications, and the exploitation of such method may bring impacts (cf. Section 6.1.4 and 6.1.5).

WAV Steganography Encoder (*WAV-Stega*): Injecting data in WAV chunk samples [116] is very similar compared to how it is injected in image pixels (as in *JPG-PNG-Stega*). Based on the WAV file format [106], this encoder checks how many *data samples* (audio samples) there are inside the data *subchunk*. For each sample (that can be composed of *X* Byte, depending on what is declared in *BlockAlign* field), the *WAV-Stega* encoder injects data in the 4 LSBs of the sample.

Text Steganography Encoder (*TXT-Stega*): The purpose of this encoder is to hide data in a text format, using Steganography. An English dictionary was used to support the *TXT-Stega*, not requiring a *File Format Sample*. Based on a sequence of bytes (data that should be hidden), the encoder generates a text output with English words. For testing purposes, the encoder has a pre-defined mapping table with word sets that corresponds to hexadecimal bytes. Since it is a bidirectional function, it is also possible to reverse and get the original input (decoding). Also, this encoder can be adjusted to generate an output of a maximum given number of bytes (i.e., a maximum number of characters).

These three steganography-related encoders were chosen since each deals with different types of media, i.e., images, audio, and text. Other types of media could be chosen, but those are a representative selection of file types that are most uploaded in the Internet nowadays [66].

4.3.2 File Format Header-related Encoders

The FileFormatHeader-related encoders use specific methods depending on the file format. The file format header (or any kind of organizational data) is explored to generate inconsistencies or to inject a high quantity of data. This encoder group is classified with a medium detection complexity, since the fields can be verified iterating throughout the file (considering a standardized file format).

The following FileFormatHeader-related encoders were developed [81]:

ID3v2 Tag Encoder (*ID3-Tag*): This encoder injects data using optional headers of an audio file format. The *ID3-Tag* uses the ID3 version 2 metadata container [93] to inject data in some fields related to the audio file, e.g., title, artist, album, track number, among others. The standard specifies a metadata tag size up to 256 MByte. Thus, this encoder uses a *File Format Sample* up to 10 seconds of audio file and builds a *X* MByte ID3v2 tag within the file. All ID3v2 fields are used by the encoder. As a parameter, it is possible to specify the amount of data to be injected in the ID3v2 fields.

Note that this encoder has the goal to exploit the lack of balance validation between the audio duration, audio quality, and file size. E.g., a MP3 or WAV file with 10 seconds and 256+ MByte cannot be considered coherent, even if generated with the best audio quality.

JPG Marker Encoder (*JPG-Marker*): This encoder takes advantage of additional JPG standardized markers to inject data inside an image file. In the JPG standard [113] is described that the marker "APPn" is application-specific, allowing vendors to store metadata. The marker is represented by the hexadecimal "FF E*n*" (2 Byte), where "*n*" can vary from "2" to "F" (hexadecimal). Therefore, it is possible to have 14 application-specific markers in one JPG file. The "APP_0" and "APP_1" are reserved to JFIF and EXIF standards, respectively. Each application-specific marker has 2 Byte to store the data length. For this reason, it is possible to have 65,535 Byte of data given the data length representation field. Moreover, there is a marker for tex-

tual comments entitled "COMM". This marker is represented by the hexadecimal "FF FE" and can carry the same amount of data as an "APP" (the marker data length representation is also 2 Byte).

The *JPG-Marker* encoder injects data in these 14 application-specific markers, plus injecting the maximum amount of data in the EXIF marker ("APP_1"). It also creates one to many "COMM" markers with data associated to it. The JFIF is left untouched, since the majority of JPG validators fully read it. A JPG file constructed by this encoder contains, at least, 1,048,560 Byte, having 983,025 Byte of the "$APP_{1..15}$" and at least 65,535 Byte of one "COMM" marker. As a parameter, it is possible to add more "COMM" markers for testing purposes.

PNG Ancillary Chunks Encoder (*PNG-Chunks*): The *PNG-Chunks* encoder injects data in non-mandatory PNG header fields. A chunk is comparable to the JPG marker: it is a data structure expressed after the PNG header [128] that conveys certain information about the image. The chunk consists of four parts: length (4 Byte), chunk type/name (4 Byte), chunk data (length in bytes), and CRC (Cyclic Redundancy Code, 4 Byte).

In order to store data, this encoder uses the chunks entitled "iTXt" and "tEXt". According to the standard both chunks store text, with one name/value pair for each – being possible to have multiple "tEXt" in the same PNG file. In fact, the data representation does not matter since the injected data in these chunks is not displayed.

Due to the 4 Byte length representation, the implemented encoder is able to inject 4 GByte of data in each specified chunk. Therefore, as encoder parameters, it is possible to specify how much data will be injected, and in which name/value pairs (it can present multiple pairs, if needed).

Email Encoder (*Email-Enc*): The *Email-Enc* encoder was implemented to explore the injection of data within the email body and attachments. The implementation is trivial, since it is an enhanced IMAP (Internet Message Access Protocol) email client. The enhanced part that differs from a simple email client is twofold. First, it takes data and transforms in a textual

representation to include in the email body (using UTF-8). Second, it also takes data and put as an email attachment. The *Email-Enc* has two parameters: the amount of data injected in an email body and in the attachment part.

These four FileFormatHeader-related encoders were chosen since each deals with different types of standards, with different types of media, i.e., images, email, and audio. Other types of standards or types of media could be chosen, but those standards are widely used to upload images, audio, and send emails in the Internet [66].

4.3.3 Appending-related Encoders

The Appending-related encoder group appends data in the end of the file in order to evaluate if Cloud services' data validation processes check end-of-file markers. This group is classified with a low detection complexity since the file size can be calculated, and end of file markers are defined in standards.

The following Appending-related encoder was developed:

Appender Encoder (*Append-Enc*): The *Append-Enc* takes a *File Format Sample* as input and appends data after its content data. Since some file formats do not have an explicit end-of-file marker (e.g., MP3), it was created a specific marker to separate the *File Format Sample* content data to the appended data. The marker is the sequence of bytes "FF FF 01 FF 02 FF 03 FF 04" (hexadecimal). The amount of data to be appended is configurable through a parameter. By default, the *Append-Enc* appends the same amount of data as contained in the *File Format Sample*. E.g., if a JPG file has 300 kByte, the encoder appends more 300 kByte of data, resulting in a file of 600 kByte. Based on the appender marker, the decoder is able to retrieve the appended data.

Although there are different manners on how to append data, e.g., appending before the end of a file or within a specific header, the Append-Enc was designed as a generic encoder to append data without modifying the original content of such file.

4.3.4 Proof-of-Concept Implementation to Test Cloud Service's Data Validation

In order to evaluate the methodology and data encoders presented in this chapter, a proof-of-concept system was implemented [81]. Such system implementation supports the execution of test cases based on the methodology described in Section 4.2. The components of the proof-of-concept system are part of the PiCsMu system, except the *Test Case Implementation* which was done specifically to perform the tests related to data validation in Cloud services.

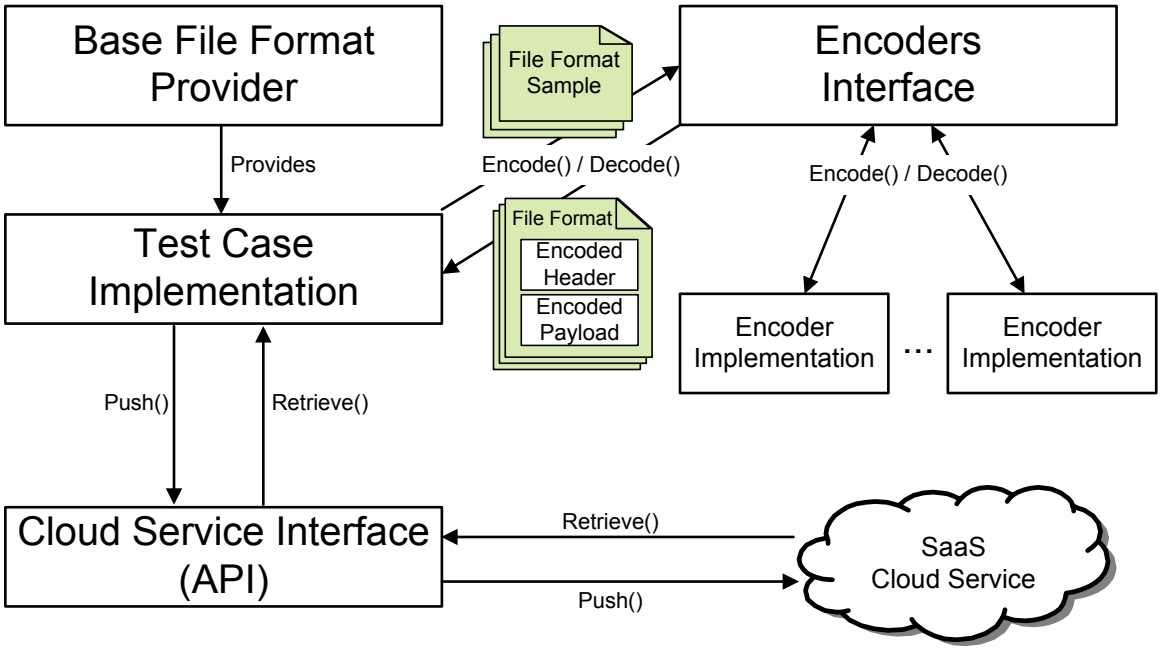

Figure 4.2: Proof-of-Concept system implementation to support the execution of test cases

Figure 4.2 depicts the architecture of such proof-of-concept system to perform the tests for evaluation purposes of Cloud services' data validation. The main component is the *Test Case Implementation*, where the test automation is done. It has the responsibility to prepare a file, push/retrieve to/from the SaaS application, and check if what was pushed is identical to what was retrieved. The *Base File Format Provider* has the responsibility to provide file samples to the *Test Case Implementation*. E.g., if observed that a certain SaaS application only accepts JPG images (Section 4.2.2, Step 1), then the *Test Case Implementation* requests to the *Base File Format Provider*

a JPG file sample. The *Base File Format Provider* selects a pre-defined file out of the specific requested type. The *Test Case Implementation* interacts with *Encoders* based on the *File Format Sample*. The *Encoders Interface* has two methods: *encode* and *decode*. The *encode* takes as input the *File Format Sample* since it needs a starting point to encode data considering a file format. The *File Format Sample* is not a strictly mandatory input parameter, as it depends on the test case. The output is a file that in terms of structure is very similar to the *File Format Sample*. However, depending on the *Encoder Implementation*, the file's organizational and content data might be modified to explore a certain fragility in the data validation process.

The *Test Case Implementation* also interacts with the *Cloud Service Interface*. This software component is service-specific, and is implemented considering chosen Cloud services to perform the test cases. For each Cloud service, a Java library that implements push and retrieve of data was used. Thus, the *Test Case Implementation* delegate the responsibility to push or retrieve the files to specific libraries that directly interact to Cloud services.

5

Recommendations based on Interactivity and Geographical Closeness

This chapter presents JSocialLib, a library that supports Social Recommendation Systems (SRS) in the task of generating recommendations, by providing two methods: interactivity- and location-based methods. Both methods are based on monitoring and collecting data from existing OSNs and provide results to SRSs that use them. This thesis focuses on two reasons for providing a tool to support a SRS in PiCsMu: (1) to expand the user base of the application – i.e., attracting more users to use PiCsMu, and (2) to enhance user's experience by recommending, e.g., to share content to specific PiCsMu friends that might be interested.

Since the PiCsMu SRS is not implemented in the context of this thesis, this chapter illustrates two use cases on how PiCsMu SRS could use JSocialLib and generate recommendations based on its method's result. Both use cases targets the expansion of the PiCsMu user base. Moreover, de-

sign and implementation details are shown in this chapter, where the API is presented, and challenges related to OSN API's rate limits are discussed.

5.1 Recommendation System Architecture

In general terms, JSocialLib can be used by applications' SRS to measure OSN interactivity and geographical closeness of users that associated their OSN identities to the given application [40]. Specifically, JSocialLib fits into the PiCsMu application architecture with the integration of one or more existing OSNs, as shown in Figure 5.1. Due to the integration of PiCsMu and existing OSNs, PiCsMu users might associate their PiCsMu identity to their existing OSN identities. PiCsMu users are able to perform such association of identities through the front-end of the PiCsMu application which, e.g., can be a Graphical User Interface (GUI). Thus, JSocialLib is able to monitor and collect data by fetching directly from OSNs, analyzing it, and providing results to the PiCsMu SRS.

The JSocialLib component, presented with stripe patterns shown in Figure 5.1, is designed, implemented, and evaluated in this thesis. The SRS, Social Core, and Front-End components interacts to JSocialLib but they are not designed nor implemented in the context of this thesis, just serving as supporting components to show the integration feasibility with PiCsMu. The PiCsMu application core component is where the file upload and download processes are implemented, and where all actions exposed to PiCsMu users are triggered – e.g., share a file or follow a PiCsMu user. Further component relations are omitted in Figure 5.1, such as, e.g., the relation between the PiCsMu application core and PiCsMu IdP, where the information about the PiCsMu OSN is kept.

The Social Core component is responsible to interact directly with OSNs for, e.g., share application-specific content, and for authentication and authorization. The association of PiCsMu application (PM-App) and OSN users' identities is done by the Social Core component. First, PiCsMu users specify in which OSN they have an OSN user account. Second, the Social Core asks permission of PiCsMu users' to allow the PM-App to directly interact with the OSN. Most OSNs use OAuth [95] for authentication and

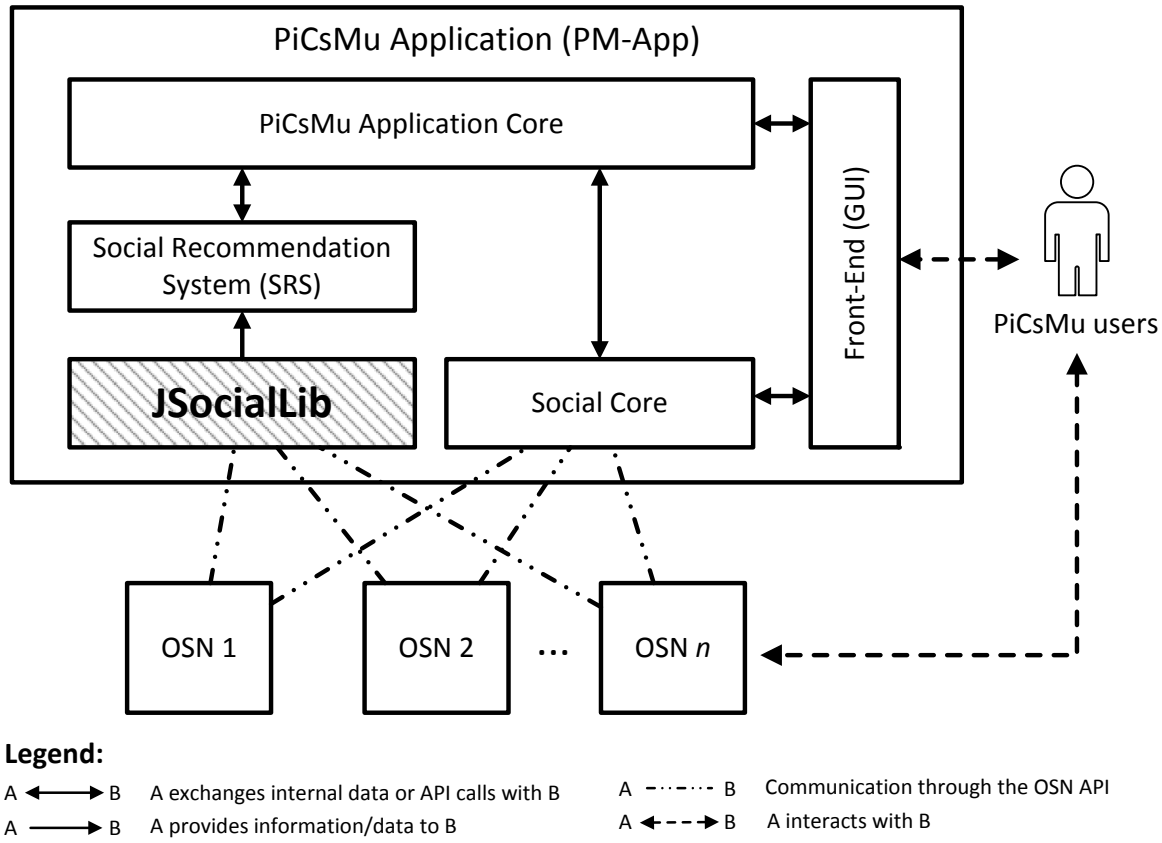

Figure 5.1: JSocialLib system architecture and interactions to PiCsMu application components

authorization of third-party applications. It is important to note that both the Social Core and PM-App Core components, which holds the PiCsMu logic, have to be aware of JSocialLib permissions' requirements to access OSN data. Such permissions can be obtained from the JSocialLib. E.g., Facebook requires a different set of permissions to access private messages, public posts, or to read OSN user's attributes. Once OSN users accept such permissions, the OSN generates an authentication and authorization token. These tokens are kept and managed by the Social Core component.

The PM-App calls the SRS, which in turn analyzes OSN data from the OSN user to infer what and to whom recommend to. The JSocialLib supports SRSs in order to reduce the implementation complexity by providing information about OSN user's interaction and geographical closeness. The SRS component calls methods provided by the JSocialLib specifying the OSN user, the authorization and authentication token, and the time frame that interactivity and geographic closeness should be observed in.

5.2 Measuring Social Network Information

This section first presents a common model [76] which JSocialLib relies to store information (Section 5.2.1). The designed common model presents attributes which are present in most well-known existing OSNs nowadays. Secondly, Section 5.2.2 details the methods to measure social data, such as the number of private messages exchanged of public posts, being based on the information available in the common model.

5.2.1 Common Model

OSNs are heterogeneous and show different concepts and social attributes. Table 5.1 provides a summary of three large OSNs (Facebook, Twitter, and Google+) showing the supported social attributes inherent to the OSN user ("Yes" and "No" for support and not supported, respectively). Each OSN has a set of concepts that differ fundamentally, e.g., Facebook's "like" and LinkedIn's "endorsing skills" that are not present in most OSNs. These differences turn the processes of collecting, monitoring, and inferring information from OSN data into challenging tasks.

Therefore, JSocialLib relies on a common model – encapsulating different concepts and different social attributes – to support the data collected from OSNs. Such common model has the characteristic to be flexible and extensible to adapt new concepts from different OSNs over time. An example of an OSN common model is the Social Data Specification released by the OpenSocial project [97], which specifies high-level entities to describe social signals and social information. Based on the Social Data Specification, Figure 5.2 depicts the common model developed here and used by JSocialLib.

The *OSNIdentity* is an abstract class representing users within an OSN, i.e., OSN users (concrete *OSNUser* class). *OSNRelationType* defines the type of a relation in an OSN (*OSNRelation* class). E.g., an OSN user can follow or be subscribed to another OSN user (e.g., as in Twitter). The "bidirectional" OSN relation type means that an OSN user must acknowledge the relationship from another OSN user (e.g., as in Facebook). *OSNLo-*

Table 5.1: Comparison of user attributes from different OSNs

OSN User's Attributes	OSNs		
	Facebook	**Twitter**	**Google+**
Age	Yes	No	Yes
Birthday	Yes	No	Yes
Current location	Yes	No	Yes
Email address	Yes	No	Yes
First/Last Name	Yes	Yes	Yes
Full Name	Yes	Yes	Yes
Gender	Yes	No	Yes
Hometown	Yes	No	No
Unique Identifier	Yes	Yes	Yes
Languages	Yes	Yes	Yes
Timezone	Yes	Yes	No
Username	Yes	Yes	Yes
Friendship/Follow Request Status	No	Yes	No

cation represents a geographic location, having a name or latitude and longitude coordinates. Such coordinates are not always present in *OSNLocation* instances, since it depends on the OSN implementation: locations within OSN user attributes, i.e., *currentLocation* and *hometown*, usually are expressed only with a unique identifier and a name (e.g., as in Facebook). *OSNContent* represents all OSN content or OSN data OSN users generate – ranging, e.g., from public posts or private messages to pictures or comments. *OSNContent* can be associated to other *OSNContent* classes to support a chain of content.

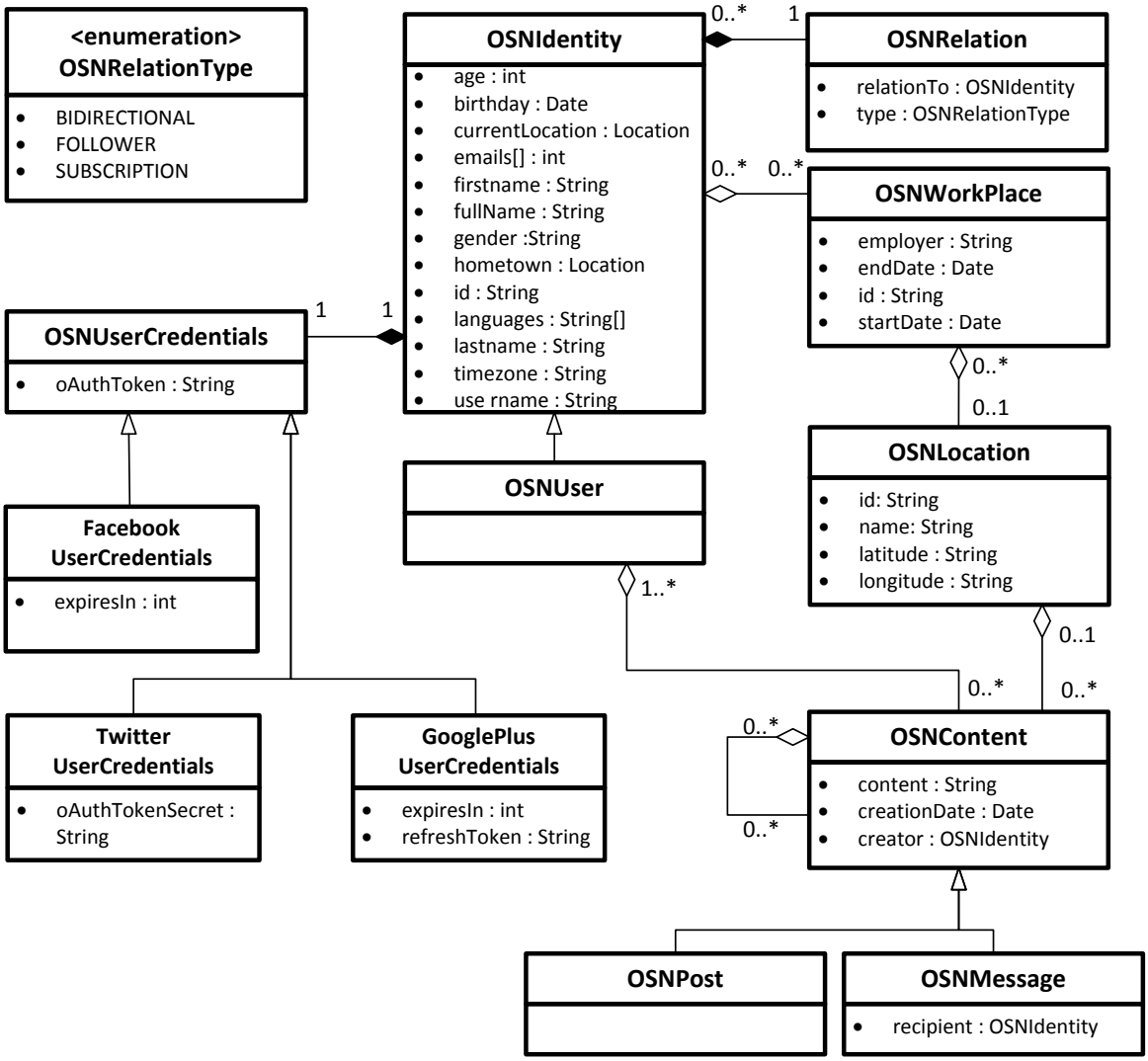

Figure 5.2: JSocialLib common model

5.2.2 Methods to Support Social Recommendation Systems

The methods presented in this section are based on a set of input parameters which are found (or can be calculated) from attributes of the common model. The given output is a ranking constituted of scores. The methods measure interactivity and geographical closeness from OSNs, presented on Section 5.2.2.1 and 5.2.2.2, respectively.

There are other methods that could also be developed [40], focusing on different OSN data. For example, re-sharing of content could be measured and quantified in a score, providing a ranking of OSN friends that most re-shares public posts from a specific OSN user. Another example is a method that compiles a ranking of OSN friends that most accept event invitations

from a OSN user. It is not possible to affirm that a combination of several different method results would lead to produce better recommendations. A complete evaluation to observe if users would accept such recommendations based on method X, Y, or Z has to be made. But, these examples, together with the interaction- and location-based methods, could provide more sources of input for SRS's algorithms, resulting of more recommendations possibilities. E.g., instead of focusing to expand the PiCsMu user base by recommending OSN friends to join the PiCsMu system (using the interactivity-based method), the PiCsMu SRS could focus on user experience enhancements by recommending to share files with OSN friends that usually re-shares, which may mean that such friend likes what a PiCsMu user shares.

This thesis has chosen to develop methods that measure interactivity and geographical location due to two reasons. The first is due to the high reflection that social interactivity has in real life: [131] concluded that social-based applications should be designed with interactions graphs in mind, of different types of OSN interaction types, since they reflect real user activity in a higher degree rather than social linkage alone, i.e., sole OSN friendship relations. Thus, based on this study, this thesis assume the higher interaction OSN users have, higher the probability that OSN users have a real friendship relation, and consequently, higher the chances that a recommendation would be accepted due to proximity and possibly trust. The second reason lies on the combination of interactivity and how geographically close OSN users are to each other. [63] shows that geographic distance strongly affects how OSN friendships are created, and spatial proximity plays a negligible role on user interactions – depending on what they interact and the platform used. Therefore, based on these assumptions from previous work, SRSs can combine both interaction- and location-based method results to decide to whom recommend and not recommend.

5.2.2.1 Interaction-based Method

The interaction-based method measures the interaction between an OSN user and his/her OSN friends, in both directions and independently: from

the OSN user to each of his/her OSN friends and from each of his/her OSN friends to the OSN user. Input parameters include: an OSN user (u), the OSN user's authentication token (at), and the observation time frame ($time$). The output is a list of OSN friends that includes the list of most interacted OSN friends whom the given OSN user has interacted with, during the specified time frame. Algorithm 1 shows the pseudo code for the interaction-based method.

Algorithm 1 Pseudo code for the interaction-based method

```
1: procedure InteractionBased(u, at, time)
2:     Array friendsScore[n], friends[n]
3:     for i=0 to i<n do
4:         friendScore[i] = IS(u, friends[i])
5:     return desc_order(friendsScore[])
```

The interaction-based method shown in Algorithm 1 outputs an array composed of each interactivity score IS (line 5) between u and his/her OSN friends ($friends[n]$), where n is the total number of OSN friends that u has. Such array is sorted in a descending order to represent a ranking output. This method considers Public Post (PP) and Private Message (PM) as the OSN interactions. Each PP or PM can be one of the following types:

Public Post Sent (PPS): A Public Post (PP) sent is a public OSN interaction of an OSN user toward another OSN user, e.g., OSN user A writing on OSN user B's profile.

Public Post Replied (PPR): It is an extension of the PPS interaction, but in the opposite direction: e.g., OSN user B publicly posting something to OSN user A, but relating to a previous PPS from OSN user A to B.

Private Message Sent (PMS): A Private Message (PM) sent is a direct message sent from an OSN user to another OSN user, which can only be accessed by the two, privately.

Private Message Replied (PMR): It is an extension of the PMS interaction, but in the opposite direction: e.g., OSN user B sending a private mes-

sage to OSN user A, but relating to a previous PMS from OSN user A to B.

These OSN interactions were chosen due to three mains reasons. First, PPs and PMs are interactions that are present in the vast majority of OSNs. However, the decision to consider only these OSN interactions does not limit a possible JSocialLib extension to encapsulate additional OSN-specific interactions. Second, previous work [92] identified that some OSN-specific interactions (e.g., Facebook "like") are not precisely perceived by OSN users. Thus, this thesis assumes, by empirically observing previous work and OSN users, that PPs and PMs represent interactions which OSN users consider most and perceive them most than other OSN interactions. Last but not least, PPRs and PMRs are considered as OSN interactions of a degree than isolated PPSs and PMSs, respectively. This empirical assumption is based on the fact that OSN interactions acknowledged by other OSN users are more perceived than unacknowledged ones. E.g., assume the following 2 scenarios: (a) The OSN user A sends n PMs to OSN user B, but OSN user B does not answer any of them, and (b) the OSN user A sends n PMs to OSN user C, but OSN user C answered $n/2$. Therefore, while comparing both scenarios, it is possible to conclude that OSN user A has a higher degree of interactivity with OSN user C than with OSN user B.

The interaction-based method does not take the content of each OSN interaction into consideration, but only the quantity of interactions. E.g., all PPSes from OSN user A to OSN user B are accounted as well as PPRs from B to A. Such approach preserves the OSN users' privacy. In order to compose the result of the interaction-based method, it is required to define the Interactivity Score (IS) in both directions. Since PPs and PMs are considered as OSN interactions, the interactivity score is defined as the sum of PPs and PMs:

$$IS(u,f) = I_{pp} + I_{pm} \tag{5.1}$$

As mentioned, this method emphasizes that OSN interactions are a balance between what is sent and what is acknowledged by the counterpart.

Thus, the ratio of PPS and PPR (R_{pp}) as well as PMS and PMR (R_{pm}) is calculated as follows, respectively:

$$R_{pp} = PPR/PPS \tag{5.2}$$

$$R_{pm} = PMR/PMS \tag{5.3}$$

In addition, lower and upper bound ratio thresholds (TL and TU) are defined, respectively, where t is a threshold value that can be adjusted depending on how strict or tolerant the method should measure the balance of interactivity:

$$TL = 1/t \tag{5.4}$$

$$TU = t \tag{5.5}$$

Therefore, if R_{pp} and R_{pm} are within the lower and upper bound ratio threshold, OSN interactions have to be considered, otherwise they are discarded. The lower and upper bound ratio threshold allows the method to identify that, e.g., an OSN user is sending messages, but the OSN friend is not answering or ignoring them. In some cases this can mean abuse or spam, which is not interactivity acknowledged by both parts.

Let S_a define the sum of a, where a represents either PPS, PPR, PMS, or PMR. ω_a defines a weight for each S_a, which determines the degree of importance for each of these OSN interactions considered, e.g., the higher the weight value on PPS, the higher importance of PPS to measure the final score of interactivity. Based on these definitions, I_{pp} and I_{pm} are defined,

respectively, while the calibration of the values ω_a and t is performed in Section 6.2.4:

$$I_{pp} = \begin{cases} \omega_{pps} S_{pps} + \omega_{ppr} S_{ppr}, & \text{if } (TL \leq (R_{pps} \text{ and } R_{ppr}) < TU) \\ \omega_{pps} S_{pps}, & \text{if } (TL \leq R_{pps} < TU) \\ \omega_{ppr} S_{ppr}, & \text{if } (TL \leq R_{ppr} < TU) \\ 0, & \text{otherwise} \end{cases} \tag{5.6}$$

$$I_{pm} = \begin{cases} \omega_{pms} S_{pms} + \omega_{pmr} S_{pmr}, & \text{if } (TL \leq (R_{pms} \text{ and } R_{pmr}) < TU) \\ \omega_{pms} S_{pms}, & \text{if } (TL \leq R_{pms} < TU) \\ \omega_{pmr} S_{pmr}, & \text{if } (TL \leq R_{pmr} < TU) \\ 0, & \text{otherwise} \end{cases} \tag{5.7}$$

5.2.2.2 Location-based Method

This method measures OSN friends that are geographically closest to a given OSN user, being based on OSN location data found either in OSN user attributes, i.e., hometown and current location, or embedded to OSN content, i.e., pictures, public posts, and private messages with location tags. Location tags represent additional information in an OSN content to express locations through GPS (Global Positioning System) [83] coordinates. Although there are other ways to obtain location data, such as read content of private messages and public posts to parse for location keywords, this method considers location data in OSN user attributes without parsing and interpreting content.

Input parameters include an OSN user (u), the OSN user's authentication token (at), and the observation time frame ($time$). The output is a list of OSN friends (referred as *LM*, location-based method) that are geographically closest to the given OSN user during the time frame specified. The location-based method is described in pseudo-code as follows:

Algorithm 2 Pseudo code for the location-based method.

```
1: procedure LocationBased(u, at, time)
2:     Array friendsScore[n], friends[n]
3:     for i=0 to i<n do
4:         friendScore[i] = LS(u, friends[i])
5:     return desc_order(friendsScore[])
```

The location-based method shown in Algorithm 2 returns an array composed of each location score *LS* (line 5) between *u* and his/her OSN friends (*friends*[*n*]), where *n* is the total number of OSN friends that *u* has. Such array is sorted in a descending order to represent a ranking output.

In order to calculate *LS*, the location-based method performs 2 steps. First, it is necessary to collect all OSN location data found for the OSN user *u* and his/her OSN friend *f*. Second, based on the found OSN location data for *u* and *f*, it has to identify if pairs of locations are closer than a certain distance threshold *c*. If the OSN location data is found in OSN user's attributes or OSN content, the distance threshold *c* is defined as c_{attr} or c_{cont}, respectively. The process to identify close locations uses the Haversine function [44], which measures the great-circle distance between two points on a sphere from their longitudes and latitudes points. Based on the Haversine distance, the value is considered as a *close match* if the output is less or equal than c_{attr} or c_{cont}. It is important to note that the close match process is performed with all pairs of locations from *u* and *f* with the same type of OSN data: OSN user attributes' pairs and location tags found in OSN content. Therefore, M_{attr} represents the number of close matches found in OSN user attributes, while M_{cont} represents the number of close matches found in OSN content. Moreover, ω_{attr} and ω_{cont} stand for weight values applied on M_{attr} and M_{cont}, respectively. As for the interaction-based method, these weight values are calibrated as shown in Section 6.2.4. Thus, *LS* is defined as presented in Equation 5.8.

$$LS(u,f) = (\omega_{attr} * M_{attr}) + \sum_{k=1}^{M_{cont}} \left(\omega_{cont} * \left|\frac{1}{D}\right|\right) \tag{5.8}$$

The location tags associated to OSN content can considerably vary depending on where OSN users are located at the moment, e.g., OSN users can post pictures or send messages while travelling. This is not the case for OSN user attributes since, e.g., current location or hometown tend to not be changed very often. In order to take such behaviour into account, D represents the time difference in seconds of when the OSN content pair was created. Thus, the longer the difference, the less the specific close match counts for the final location score. Figure 5.3 illustrates an example calculating LS for OSN user A, which has friends B and C. Locations X and Y are already identified as a close match. OSN user A was at location X on day d. OSN user B was at location X on d-10 and again on d-3, as well as at location Y at d-2. OSN user C was at location X at d+1. In this example, the method accounts a higher location score between A and C. The reason is that OSN user A and C have a close match with only one day of difference, while OSN user A and B have a close match with 10, 3, and 2 days of difference.

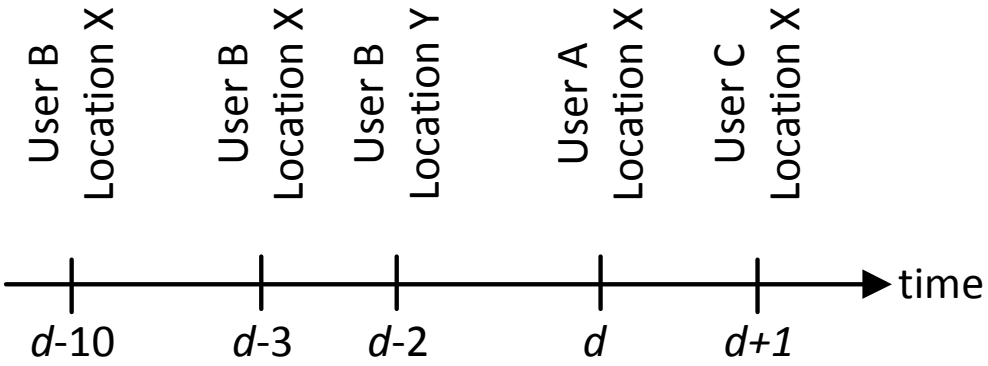

Figure 5.3: Location-based method example

5.3 Use Cases: Recommendations for PiCsMu Users using JSocialLib

As stated in Section 1.4, one of the advantages of social recommendations is to extend the social graph of a given social network. The two methods presented in Section 5.2.2 provide support to the PiCsMu SRS, giving a powerful instrument (i.e., accurate results from user's perspective) to generate social recommendations. Therefore, two use cases, in which the PiCsMu SRS

applies JSocialLib methods, enable the extension of PiCsMu social network graph by performing the following recommendations:

- *Use Case 1 (UC1):* Recommend new PiCsMu friends to a PiCsMu user X, based on PiCsMu user X's friends from existing OSNs.
- *Use Case 2 (UC2):* Recommend the adoption of PiCsMu system to OSN friends of PiCsMu user X.

While UC1 depends on both requirements, UC2 only depends on requirement 1:

- *Requirement 1 (RE1):* PiCsMu user X should have at least one OSN identity associated to his/her PiCsMu identity.
- *Requirement 2 (RE2):* At least one of the OSN friends (e.g., OSN friend Y) of the associated OSN identity from PiCsMu user X should be using the PiCsMu system and also has an OSN identity associated to a PiCsMu identity.

UC1's goal is to recommend new possible friends to a PiCsMu user inside the PiCsMu social network, assuming that they are friends in existing OSNs. In order to illustrate UC1, consider Alice and Bob as two PiCsMu users, and both being friends in Facebook OSN – in which they interact a lot. Moreover, both have associated their Facebook identities to their respective PiCsMu identities. However, they don't know that each other use the PiCsMu system. In this use case, the PiCsMu SRS can use JSocialLib to get the Facebook friends that Alice most interact to and, thus, check which ones of these Facebook friends also have an PiCsMu identity. Those which Alice interacts a lot on Facebook *and* also have PiCsMu identities will be recommended to Alice as potential new friends in the PiCsMu social network.

UC2's goal is to recommended the adoption of the PiCsMu system to individuals out of the PiCsMu social network, targeting those that have ties to a PiCsMu user in an existing OSN. In order to illustrate UC2, consider Alice and Bob. The former is a PiCsMu user with an Facebook identity

associated to its PiCsMu identity. The latter just holds a Facebook identity, being a friend of Alice in Facebook OSN. In this scenario, the PiCsMu SRS can use JSocialLib to get the Facebook friends that Alice most interact to and, thus, recommend the adoption of the PiCsMu system to a selection of Alice's Facebook friends.

In both use cases UC1 and UC2, the interaction- and location-based methods can be combined in the following manner: instead of only generating recommendations to friends that a given PiCsMu user X interacts to, the PiCsMu SRS can generate recommendations to whom PiCsMu user X most interacts to *and* is geographically closest to. Combining both methods depends on whom the SRS aims to target on such recommendations, or also depends on the acceptance of previous recommendations. For example, the SRS can start generating recommendations based on both methods and, at some point in time, check how is the acceptance. If the recommendation's acceptance is lower than a certain threshold, the SRS would use both methods in a separate manner or even with different weights.

5.4 Implementation

A Java-based prototypical implementation of JSocialLib was developed. It was built as a JAR (Java Archive) package exposing a set of Java methods to the PiCsMu SRS.

5.4.1 JSocialLib API and Supported OSN Providers

JSocialLib defines a *OSNProvider* class, which is the interface to specific OSN provider's implementation. JSocialLib has three concrete OSN provider implementations: *FacebookOSNProvider*, *TwitterOSNProvider*, and *GooglePlusOSNProvider* classes. Each of the concrete OSN providers, respectively, uses the following third-party libraries to communicate to the respective OSN: RestFB [104], Twiter4J [121], and Google Plus Java API client [47]. More OSNs can be supported by extending the *OSNProvider* class interface with specific OSN implementation classes. The *OSNProvider* implementation exposes a set of

methods that are used by the JSocialLib interaction- and location-based methods for a given OSN user. E.g., *OSNProvider* methods are `countPrivateMsgs(Date d1, Date d2)` that counts private messages between a time interval, `getContentBetweenDate(Date d1, Date d2)` that fetches all OSN content found between a time interval, `getCurrentLocationAttribute()` that returns the current location OSN attribute, `getRelations()` which retrieves all the OSN relationships, amongst others.

Moreover, JSocialLib defines the *JSocialLibAPI* class, which is the concrete implementation exposing the following Java methods: `getFriendsMostInteractTo()` implementing the interaction-based method described in Section 5.2.2.1, and `getFriendsGeographicallyClosestTo()` implementing the location-based method (cf. Section 5.2.2.2). Both *JSocialLibAPI* class methods receive as parameter a `String oAuthToken` representing the OAuth token of a given OSN user, and two date objects forming the time frame in which these methods should fetch OSN data to draw results. Tasks inside each of these methods, e.g., `countPrivateMsgs()` and `getRelations()`, are parallelized to achieve a better execution time. In addition, the observer/observable design pattern is employed to support asynchronous results.

5.4.2 OSN API's Rate Limits

A considerable challenge faced by JSocialLib is to find when to fetch OSN information for analysis. Thus, the limitations of API requests of OSNs had to be taken into consideration for materializing a feasible JSocialLib implementation [75]. E.g., Facebook and Twitter limit API calls to a certain rate per OSN user, and also per OSN application. E.g., Facebook has the limit of 600 calls per 10 minutes per OSN user authentication token. Also, there exists an application limit regarding CPU and memory consumption of Facebook's servers. The exact limit per application is not fully disclosed, only that the resource consumption limit is "a percentage of the monthly active OSN application users" [37], i.e., if the OSN application does not

have users, it can fetch less OSN information. If the OSN user authentication token limit or the OSN application limit is reached, API requests are blocked for at least one hour, which can be critical for JSocialLib. Twitter only limits access per OSN user authentication token and has transparently documented rate limits. JSocialLib address this situation by employing an adaptive waiting time upon the detection of an API call error. It is not possible to distinguish the error's type, especially to detect if the API rate limit has been reached or not. However, it is possible to detect network problems, e.g., if JSocialLiB could not connect to the API's endpoint port. If any other error that is not inherent to the network happens, JSocialLib applies a waiting time that increases by 5 minutes every time that an error is returned.

Another challenge faced is the amount and accuracy of data returned by OSN API calls. Facebook, e.g., limits to obtain some of its OSN users data (e.g., stream public messages) only retrospectively to 30 days or maximally 50 posts, whichever is greater. Moreover, the data returned by Facebook may be not consistent, i.e., executing the same API call on two different days may reveal different data – even though the Facebook data for such specific OSN user does not change. Related to such accuracy limitations, a bug report was filed about unexpected behavior [38] during the development of this work. The bug was confirmed and set to be further investigated. In order to decrease that impact of lack of accuracy and the lack of data, JSocialLib applies weights within these two methods.

6 Evaluation

The PiCsMu system is evaluated following a bottom-up approach. First, single pieces of the system were evaluated to show the feasibility of exploring Cloud services' data validation and its impacts, as well as the accuracy of the JSocialLib library to support the PiCsMu SRS.

Second, the end-to-end PiCsMu system was evaluated concerning scalability and performance, by measuring the data overhead, the index overhead, and times to upload/download files. Lastly, a legal discussion is presented to point what are the possibly affected regulation areas with the technology employed by the entire PiCsMu and, more generically, Cloud storage overlay systems.

6.1 Data Validation in Cloud Services

Following the methodology presented in Section 4.2, a set of test cases were developed. These test cases were applied to public Cloud services, following the selection criteria based on popularity. The test cases described in

this section are directly attached to the Test Case Implementation component, part of the proof-of-concept system implementation presented in Section 4.3.4. Results and a discussion about impacts are shown in the end of this section.

6.1.1 Cloud service's Selection Criteria

The Cloud services are selected based on popularity. This thesis does not focus on performance, cost, security, ease-to-use, and other various criteria that could be applied to select Cloud services to be tested. The reason to exclude these criteria and focus on popularity is that widely popular Cloud services tend to present a better-defined data validation process, while, in comparison, less popular Cloud services tend to present a less well-defined process.

This thesis assumes a higher probability that the data validation process of popular Cloud services is refined over time – due to the high amount of data being pushed, and, therefore, more malformed data reported as not valid. In this scope, popularity of Cloud services is empirically measured observing, e.g., number of users, volume of data pushed, and the Alexa ranking [3].

6.1.2 Chosen Cloud Services

A list of Cloud providers was compiled based on the Alexa ranking: it was chosen the best ranked Cloud services, offering a SaaS platform with generic-data restrictions, with the higher number of users, and the higher volume of data pushed per month/year to their services. Based on such selection, the chosen Cloud services, as well as the restrictions of each service are presented in Table 6.1. Note that even if restrictions can change at any time, those restrictions are the ones which were considered during the test cases' design phase. Table 6.1 was compiled according to Step 1 of the presented test methodology in Section 4.2.2, and selection criteria in Section 6.1.1.

Table 6.1: Selection of Cloud providers and Cloud services, including their restrictions

Provider	Service	Restrictions
Google	Picasa	File Formats: JPG, TIFF, BMP, GIF, PSD, PNG, TGA. Images up to 800x800 pixels are not taken into consideration to the free storage (1 GByte). Images beyond 800x800 and up to 2048x2048 are stored taking space within the 1 GByte. File size: up to 20 MByte to upload.
	GMail	Any text format in the email body. Any File Format as Attachment. The email should be up to 25 MByte.
	Google+ Status Update	Up to 100,000 characters.
	Google Docs	Documents: 1,024,000 characters. Uploaded document files that are converted to Google format should be up to 2 MByte.
Facebook	Status Update	63,206 characters.
	Photos	File formats: GIF, JPG, PNG, PSD, TIFF, JP2, IFF, WBMP, XBM. Up to 720x720 pixels (display). Up to 2048x2048 pixels to upload (but being resized afterwards). Album with a limit of 200 photos.
SoundCloud	Sound Cloud Audio	File Formats: WAV, OGG, MP2, MP3, or WMA. Up to X minutes of audio. In the free account, up to 120 minutes.
TwitPic	Image Service	File Formats: GIF, JPG, and PNG. Images up to 10 MByte to upload.
Imgur	Image Service	File Formats: JPG, GIF, PNG, etc. Most files are converted to PNG after the upload. Up to 10 MByte to upload. However, if the image is over 1 Mbyte then it will automatically be compressed or resized to 1 MByte.
ImageShack	Image Service	File Formats: JPG, PNG, ICO, BMP, and TIFF. File size up to 5 MByte to upload (otherwise resized).

6.1.3 Test Cases

All test cases follow the test method presented in Section 4.2.2. Step 2 of the test method describes the construction of data using different encoders. Therefore, test cases in this section specifically describe how the encoder is applied to specific services with different parameters (e.g., varying the used *File Format Samples* or the quantity of data injected). Based on the test method, implemented encoders, and chosen Cloud services, the following 25 test cases are described:

1. Encoder: *JPG-PNG-Stega*. Target: Google Picasa, TwitPic, Facebook (FB) Photos, Imgur, ImageShack. Unique test.
2. Encoder: *WAV-Stega*. Target: SoundCloud. Unique test.
3. Encoder: *TXT-Stega*. Target: Facebook Status Update, Google+ Status Update. Unique test.
4. Encoder: *ID3-Tag*. Target: SoundCloud. Test Cases:
   - 4.a. ID3v2 tags with 50 MByte of data.
   - 4.b. ID3v2 tags with 256 MByte of data.
   - 4.c. ID3v2 tags with 300 MByte of data.
5. Encoder: *JPG-Marker*. Target: Google Picasa, TwitPic, Facebook Photos, Imgur, ImageShack. All test cases consider the aforementioned 14 "APPn" marker. For each target, the following test cases are performed:
   - 5.a. JPG with 1 "COMM" marker.
   - 5.b. JPG with 5 "COMM" markers.
   - 5.c. JPG with 10 "COMM" markers.
   - 5.d. JPG with 50 "COMM" markers.
   - 5.e. JPG with 100 "COMM" markers.
6. Encoder: *PNG-Chunks*. Target: Google Picasa, TwitPic, Facebook Photos, Imgur, ImageShack. For each target, the following test cases are performed:
   - 6.a. 1 name/value pair as "tEXt" with 250 kByte, and "iTXt" with 250 kByte of data.

6.b. 5 name/value pairs as "tEXt" with 250 kByte each, and "iTXt" with 250 kByte of data.

6.c. 10 name/value pairs as "tEXt" with 250 kByte each, and "iTXt" with 250 kByte of data.

6.d. 20 name/value pair as "tEXt" with 1 MByte of data each, and "iTXt" with 1 MByte of data.

6.e. no "tEXt", and "iTXt" with 1 MByte of data.

7. Encoder: *Email-Enc*. Target: Google GMail. Test cases:

   7.a. Email body: 25 MByte; no attachment.

   7.b. Email body: 25 MByte; Attachment: 1 MByte.

   7.c. Email body: 10 MByte; Attachment: 15 MByte.

   7.d. No email body; Attachment: 5 MByte.

   7.e. No email body; Attachment: 26 MByte.

8. Encoder: *Append-Enc*. Target: Google Picasa, Google GMail, Google Docs, SoundCloud, TwitPic, Facebook Photos, Imgur, ImageShack. For each target, use the *Append-Enc* encoder to:

   8.a. Append the double of the *File Format Sample* size.

   8.b. Append 1 MByte of data.

   8.c. Append 10 MByte of data.

   8.d. Append 100 MByte of data.

6.1.4 Results

These test cases were performed and generated the results shown in Tables 6.4, 6.2, and 6.3. Within table cells, "N/A" stands for "Non-Applicable", meaning that a test case did not target that specific Cloud service; symbol "✔" shows that the test case was successful, meaning positive results in test methodology Step 3, 4, 5, and 6 (cf. Section 4.2.2); and symbol "✘" means that test methodology Steps 3, 4, 5, or 6 were unsuccessful.

Test Cases 1, 2, and 3 were successful for the proposed targets. For Test Case 1, the image file restrictions were respected, since Steganography-related encoders would be unsuccessful due to Cloud services' image resizing after upload. Test Case 4.c was unsuccessful due to the fact that the

pushed MP3 file does not follow ID3v2 standards. The Cloud service did not allow the audio file to be pushed (unsuccessful in Step 4). Test Case 5 demonstrated to have different results depending on the target. While Imgur accepted all files and also made them available to be retrieved in its original form, other Cloud services denied the upload (unsuccessful in Step 4). The unsuccessful test cases targeting Google Picasa, TwitPic, and Facebook Photos resulted in error messages such as "there was a problem with the image file" or "file must be an image or video". The failed test case targeting ImageShack resulted in a "file too big to be uploaded" error, since the total file size surpassed Cloud service restrictions.

Table 6.2: Results of Test Cases 1, 5, and 6

Test Cases		Cloud services				
		Google Picasa	TwitPic	FB Photos	Imgur	ImageShack
1		✔	✔	✔	✔	✔
5	a	✔	✔	✘	✔	✔
	b	✔	✔	✘	✔	✔
	c	✔	✔	✘	✔	✔
	d	✘	✘	✘	✔	✔
	e	✘	✘	✘	✔	✘
6	a	✘	✔	✘	✔	✔
	b	✘	✔	✘	✘	✔
	c	✘	✔	✘	✘	✔
	d	✘	✘	✘	✘	✘
	e	✘	✔	✘	✘	✘

Test Case 6 presented different results due to the following reasons. When Picasa was targeted, it seems that the data validation takes into consideration the existence of ancillary chunks, mainly "iTXt" and "tEXt" (unsuccessful in Step 4). When tried to perform an additional test case, where a PNG file was uploaded with another type of ancillary chunk (containing

1 MByte of data), the test was successful. Facebook accepted to push the encoded files, but when they were retrieved they did not contain the original data (unsuccessful in Step 6). For the other targets, Test Case 6 was unsuccessful when the Cloud service had upload file size restrictions (unsuccessful in Step 4). E.g., ImageShack just accepts 5 MByte, TwitPic 10 MByte, and Imgur 1 MByte. Moreover, Imgur resized the original files after the upload and therefore Test Cases 6.b, 6.c, and 6.d were unsuccessful (Step 6): the retrieved file was not the same as the original.

Table 6.3: Results of Test Cases 2, 3, 4 and 7

Test Cases		Cloud services			
		SoundCloud	Google GMail	Google+ Status Update	FB Status Update
2		✔	N/A	N/A	N/A
3		N/A	N/A	✔	✔
4	a	✔	N/A	N/A	N/A
	b	✔	N/A	N/A	N/A
	c	✘	N/A	N/A	N/A
7	a	N/A	✔	N/A	N/A
	b	N/A	✘	N/A	N/A
	c	N/A	✔	N/A	N/A
	d	N/A	✔	N/A	N/A
	e	N/A	✘	N/A	N/A

Within Test Case 7, GMail demonstrated to be coherent to the employed restrictions, not presenting data validation issues. However, combining the Email-Enc with Steganography-, FileHeaderFormat-, and Appender-related encoders, it was possible to retrieve hidden information persisted in email body and attachment files.

Test Case 8 demonstrated to be successful even though it is the most trivial encoder to detect and prevent. For Picasa, GMail, Docs, TwitPic, Imgur, and ImageShack, Test Cases 8.d and 8.c was unsuccessful due to the upload

file size restriction and not due to the used encoder. Thus, the encoded file was not even accepted for upload (unsuccessful in Step 4). Facebook Photos accepted to upload the files. But within Step 5 and 6 the data validation transformed the image files and cut the appended data.

Table 6.4: Results of Test Case 8

Test Cases		Cloud services							
		Google Picasa	Google Docs	Google GMail	SoundCloud	TwitPic	FB Photos	Imgur	ImageShack
8	a	✔	✔	✔	✔	✔	✘	✔	✔
	b	✔	✔	✔	✔	✔	✘	✔	✔
	c	✔	✘	✔	✔	✔	✘	✘	✔
	d	✘	✘	✘	✔	✘	✘	✘	✘

6.1.5 Discussion of Impacts

The importance of discussing impacts of test cases outcome lie on the fact that data validation weaknesses might be explored in a large scale. If impacts are not assessed and understood beforehand, possible solutions for ill-defined data validation rules might not be effective.

Although other impacts could be discussed, e.g., privacy and reliability, this discussion concentrates on security as well as accounting and charging. The reason to choose these two impact areas is that, during the test executions, the tests revealed two conditions: (1) the possibility of users downloading malicious or illegal shared files triggered by test cases, but perceived as normal files; and (2) the possibility of malicious users storing more resources than allowed (or under normal operations) in one single Cloud service account. Moreover, both impacts have directly influences on Cloud providers' business – higher the security and accounting/charging impacts, the higher chances of revenue losses.

6.1.5.1 Security Impacts

The performed tests identified a considerable security impact: unawareness of persisted content. This is related to security since Cloud services may distribute data (e.g., by sharing a picture, or turning public an audio file) that they do not filter and are not aware of. Cloud providers may face legal issues due to the distribution of, e.g., illegal content under the jurisdiction where the data is persisted. Nowadays Cloud providers have multiple datacenter facilities around the globe, employing a Content Delivery Network (CDN) to distribute/replicate data where is appropriated (based on, e.g., content rules). If these content rules do not take the injected data into consideration, CDNs are prone to take unappropriated distribution/replication actions. A typical example to illustrate this impact can be observed if a user persists copyrighted software into a Cloud service and publicly share it – assuming people know how to decode and get the injected data out of the encoded file. Malicious users can benefit from Cloud services' unawareness to spread content, e.g., using the PiCsMu system.

As long as data validation processes remain ill-defined, and malicious users exploit it in a large scale, other security impacts as application failures and Denial-of-Service may be observed. E.g., after the data validation, the SaaS application may process the file to display in the Web site, or add functions as image editing. If the whole process is not consistent, the Cloud service can suffer faults due to unexpected organizational and content data that bypassed the data validation process.

6.1.5.2 Accounting and Charging Impacts

In two test cases, the chosen Cloud services suffered impacts in their accounting and charging systems. At the time this evaluation was performed, Google Picasa provided a free account with 1 GByte of storage. Table 6.1 shows that "images up to 800x800 pixels are not taken into consideration to the free storage (1 GByte), and images from 800x800 up to 2048x2048 pixels are stored consuming space within the 1 GByte space". The Test Cases 1, 5, and 8 used images with a resolution below 800x800, injecting data, but not enhancing the *File Format Sample* resolution. After uploading the en-

coded image files, the account space consumption remained with 1 GByte, even if the encoded images being very large. This means that, by exploiting the data validation process, users are able to store any data without being accounted and charged for – since the free 1 GByte of storage remains unused. Therefore, even with with Google charging plans for additional storage, it does not become necessary to subscribe to a plan to obtain more storage space when using the data encoders from Test Cases 1, 5, and 8.

At the time this evaluation was performed, SoundCloud provided a free account with 120 minutes with audio storage. Test Case 4 is able to store MP3 audio files with 10 seconds and 256 MByte. An MP3 file, with sample rate of 44,100 Hz, bit rate of 128 kbit/s, no ID3v2 tags, and 10 seconds, consumes on average 150 kByte. A free account is intended to store 720 MP3 files with 10 seconds each, consuming 105 MByte; however, by employing the *ID3-Tag* encoder, it was possible to consume up to 180 GByte with the same amount of files. According to the present SoundCloud charging scheme [109], a "pro" account costs USD 35 per year, and let users to store up to 360 minutes of audio (3 times more than the free account). Considering a "pro" account, it is possible to store 2160 MP3 files with 10 seconds and no ID3v2 tags, consuming 324 MByte. In this case, the amount of data stored in normal conditions using a "pro" account is considerably less than using the data encoder from Test Case 4, and store data with the free account. If a user wants to store 180 GByte of data in SoundCloud, but without using the encoder used in Test Case 4, it would be necessary 555.5 "pro" accounts, totalling USD 18,887 per year.

In both cases a considerable additional amount of data may be stored in Cloud services' servers without additional charges – since data volume is not accounted and not considered by the data validation process.

6.2 Recommendations based on Interactivity and Geographical Closeness

The prototyped JSocialLib library, with its two interaction- and location-based methods, had been evaluated by observing if the methods are accurate according to users' perception of OSN friends' interactivity and geo-

graphical closeness. The evaluation does not include JSocialLib library integrated into the PiCsMu system. The reason is to isolate JSocialLib results and concentrate on how the library performs without having interferences – thus, not evaluating the SRS itself, but the input quality given from JSocialLib to the SRS.

Isolating the evaluation provides an individual understanding of what is JSocialLib impact to SRSs' implementation. The higher accuracy of results given by JSocialLib means a higher probability that PiCsMu users would accept recommendations when the JSocialLib is integrated to the PiCsMu SRS, regardless of the algorithm that the SRS uses.

6.2.1 Methodology

The JSocialLib evaluation consists of 4 main steps: (1) data collection, (2) hypothesis compilation, (3) methods' calibration, and (4) evaluation results (cf. Section 6.2.2, 6.2.3, 6.2.4, and 6.2.5).

Step 1 collects the data of real OSN users from different nationalities, which was achieved through a Web survey. The OSN data collected was randomly divided into two equal portions: while the first portion is used in the calibration Step 3, the second portion is applied to execute the calibrated methods, especially to evaluate results. Step 2 compiles the hypothesis based on experimental and exploratory procedures. The first portion of the collected data was manually explored to gain insights of what the average results for the JSocialLib methods' evaluation could be. JSocialLib methods are calibrated in Step 3 to determine the best set of parameters (weights and threshold values). A vast amount of parameter sets are generated and statistics computed for each of the two JSocialLib methods, mainly with the goal to calibrate the method to perform as close as possible with OSN users' perceptions. Step 4 uses the calibrated JSocialLib methods and executes them on the second portion of the OSN data collected.

6.2.2 Data Collection

The OSN data collection was performed via a survey for Facebook users, since Facebook has the largest amount of monthly active users for an evaluation.

Table 6.5: Survey questions

Question Group	Question Number	Question
Interaction-based	1.1	How often do you interact with Facebook friends?
	1.2	Who are the top 5 Facebook friends you most interacted with during the past 6 months?
	1.3	How do you rank those selected Facebook friends based on the amount of interaction in a descending order?
Location-based	2.1	How often do you specify or attach your current location when posting on Facebook?
	2.2	Who are the top 5 Facebook friends you were the geographically closest to during the past 6 months?
	2.3	How do you rank those selected Facebook friends based on geographical closeness in a descending order?

The survey Web site `http://social.csg.uzh.ch` runs a software written in JavaScript and PHP acting as a Facebook application [75]. Visitors have to enable cookies and ensure that no pop-up blocker software is active in their Web browser. On one hand, the survey simulates an application embedding existing OSNs into their solutions. On the other hand, the survey did not act as an application for serving content, but to obtain permissions from OSN users to collect their OSN data and to obtain their

perception about these methods developed. In order to store survey answers and OSN data collected of each user, a MySQL database was used. The survey's Web site was optimized for both desktop browsers and mobile devices.

Within the first step of this survey the OSN authentication and authorization was performed, where each participant authenticates its Facebook identity and accepts permissions for the survey Facebook application. The result of this step is the authentication and authorization token (OAuth token) stored in the MySQL database, under users consent. The second step followed by answering questions posed. The two main questions with three sub questions for both of them are shown in Table 6.5.

The first three questions (1.1 to 1.3) are related to the interaction-based method and the last three questions (2.1 to 2.3) are related to the location-based method. Questions 1.1 and 2.1 measure the frequency the participant interacts with or uses locations' tags on Facebook. The fixed answer set allows for either *frequently*, *occasionally*, or *never* as possible answers. Answers for questions 1.2 and 2.2 are given by selecting Facebook friends, with an interface to enable fast searches. Question 1.2 captures the OSN user perception regarding the user's interactivity, while question 2.2 captures the user perception regarding the geographical closeness of friends. In question 1.3 and 2.3 the participant has to rank those friends selected in a descending order; the first position defines the friend that the participant most interacted with or was geographically closest to, respectively. The outcome of questions 1.2, 1.3, 2.2, and 2.3, is a list of 5 OSN friends for each participant of the survey, ranked in a descending order.

In total 290 participants completed the survey, collected during 30 days. Table 6.6 presents the demographics related to the gender and age of all survey participants. Moreover, the survey Web site had visits from 20 countries and survey answers are from 9 countries.

Since the survey asks questions from a retroactive period of six months, those results generated a considerable amount of OSN data to be collected for each user, even though the actual content of an interaction or location is not stored. In order to handle this amount of data to perform evaluations with, the Facebook data of each user is cached in a MySQL database. The

Table 6.6: Survey demographics related to gender and age

Age	Gender	
	Male (Total: 52.7%)	**Female (Total: 47.3%)**
13-17	0.58%	0.30%
18-24	12%	16%
25-34	31%	25%
35-44	6.1%	3.5%
45-54	1.2%	0.60%
55+	1.9%	1.9%

database consists of three tables: relation, interaction, and location. The table relation caches OSN users' own attributes and corresponding attributes of each of the user's OSN friends. The interaction table caches all interactions of each OSN user in the past six months, counting back from the day the survey was submitted. Finally, the table location caches all locations found for each OSN user in the same time period. The caching of OSN data became necessary since it is the only way to prevent the survey application being blocked by Facebook (due to API calls rate limits, as discussed in Section 5.4.2). Moreover, the caching process enables the calibration of JSocialLib methods' testing with more parameter sets, since the results' computation was performed locally without the need to perform an API call to Facebook. Table 6.7 shows the summary of the collected amount of OSN data.

Table 6.7: Summary of the collected amount of OSN data

Table	Size (MByte)	Number of Rows
relation	108.3	122,439
interaction	63.7	123,454
location	102.8	244,885

6.2.3 Hypothesis

A manual analysis of the survey was made over 10 survey results: the JSocialLib was executed with the survey results' input and the results were empirically analyzed observing how the interaction- and location-based methods performed. Based on these empirical analysis, the following hypothesis have been defined by computing an average of method's outputs:

> **Hypothesis.** Both interaction- and location-based methods are able to estimate at least 2 out of 5 friends that reflect OSN users' perceptions.

6.2.4 Method's Calibration

Each JSocialLib method depends on a set of parameters: weights (ω_n), distance thresholds (c_{attr} and c_{cont}), and interactivity ratio threshold (t). The calibration of the methods has to find the best set of values for these parameters. In addition, as many parameters sets as possible have been tested for each method. Table 6.8 summarizes those parameters calibrated for both methods as well as presenting the interval of values assigned to each. The value for the t parameter is fixed to 2, 3, and 4, since the ratio difference from 2 to 4 is large enough, when applied to the lower and upper threshold equations (cf. Equations 5.4 and 5.5, respectively).

The value interval of $[1;9]$ for all parameters (except t) are divided into two independent groups: 1, 3, 5, 7, 9 and 2, 4, 6, 8. The generation of possible parameters sets is a combination of the value intervals for each method parameter, in an independent manner. Thus, e.g., within interaction-based method, the parameters ω_{pps}, ω_{ppr}, ω_{pms}, ω_{pmr} should be combined considering the set 1, 3, 5, 7, 9 of possible values, which give the permutation of $5^4 - 5$ possible values for each 4 parameters. This results in 625 parameter sets. The combination of the set 2, 4, 6, 8 for the same parameters give the permutation $4^4 - 4$ possible values for each 4 parameters. This results in 256 parameter sets. Besides the two groups of value intervals, it is important to note that the parameter t has 3 possible values, 2, 3, or 4. Therefore, summing 625 and 256 parameter sets, each having 3 possible t

Table 6.8: Parameter set for each JSocialLib method

Method	Parameter	Description	Interval Value
Interaction-based	ω_{pps}	Weight for Public Post Sent	[1;9]
	ω_{ppr}	Weight for Public Post Received	[1;9]
	ω_{pms}	Weight for Private Message Sent	[1;9]
	ω_{pmr}	Weight for Private Message Received	[1;9]
	t	Interactivity ratio threshold	[2,3,4]
Location-based	ω_{attr}	Weight for locations found in OSN user's attributes	[1;9]
	ω_{cont}	Weight for locations found in OSN content	[1;9]
	c_{attr}	Distance threshold for locations found in OSN user's attributes	[1;9]
	c_{cont}	Distance threshold for locations found in OSN content	[1;9]

values, results in 2,643 parameter sets generated for the interaction-based method. The same process applies to the location-based method, considering the 4 parameters, resulting in 881 parameter sets. If a full combination of 9 different possible values is used for each parameter in both methods, it would be generated 59,049 and 6,561 parameter sets for the interaction- and location-based methods, respectively. The approach of dividing the value intervals into two independent groups of possible values generates less parameter sets. Although such approach increases the granularity to find the best possible values for each method, the calibration is performed

in a faster manner still testing values in both extremes within the interval, e.g., large parameter values for some parameters, combined to very small parameter values for other parameters.

For the calibration, each JSocialLib method is executed with all corresponding parameter sets for half of the collected OSN data set (randomly chosen). The process generates 383,235 interaction-based method runs and 127,745 location-based method runs to compute. The calibration was performed on a high-end server machine with 24 CPU cores, SSD hard disks, and 64 GByte RAM, which is possible to complete 1,400 interaction-based and 700 location-based runs per minute, respectively. The time taken to run the calibration is important since adaptations to the methods may occur over time and, thus, a new generation of possible parameters is required in a reasonable time frame. In this case, it took 273.7 and 182.4 minutes to run the interaction- and location-based methods for all parameter sets, respectively.

In order to compare the JSocialLib methods' results applying the values of each generated parameter set, two metrics are defined: *rank difference* and *match count*. These metrics are chosen since JSocialLib is evaluated by observing accuracy to user's perception and requires metrics to compare how the ranking positions differ, as well as how many ranking entries fully match. The metrics compare the OSN user perception (answered in the survey) to the JSocialLib methods' results. The metric values are calculated based on the parameter sets generated. The Cosine Similarity [108] metric was considered. However, since it excludes the difference of each OSN friend rank by only computing vector distance, this thesis opted to not use it.

Rank Difference: This metric calculates the total difference in ranking positions between the ranked list of 5 friends answered in the survey, to the JSocialLib methods' result for each generated parameter set. Table 6.9 illustrates in an example how the rank difference metric is calculated. This is done for all parameters sets and the mean of total rank differences is computed (*rank difference mean*).

Match Count: This metric counts the matches of rank positions. If the rank position is the same for the OSN user answer and in the JSocialLib

Table 6.9: An example of the rank difference metric

OSN Friends	User Answer's Ranking	JSocialLib Result Ranking	Rank Difference
Friend A	1	53	52
Friend B	2	8	6
Friend C	3	1	2
Friend D	4	15	11
Friend E	5	3	2
		Total Rank Difference:	**73**

result, it is counted as an *exact match count*. If an OSN friend is ranked within any of the first 5 rank positions in the OSN user survey answer and in the JSocialLib method result, it is counted as an *top5 match count*. Table 6.10 shows an example of how to calculate the match count metric. The OSN friend B is counted as an *exact match* since the OSN user answer and the JSocialLib have the same ranking value (2). The OSN friends B, C, and E are counted as a *top5 match* since OSN user answer and JSocialLib result ranking values are equal or below 5.

Table 6.10: An example of the match count metric

OSN Friends	User Answer's Ranking	JSocialLib Result Ranking
Friend A	1	20
Friend B	2	2
Friend C	3	5
Friend D	4	15
Friend E	5	3
		Top5 Match Count: 3
		Exact Match Count: 1

Based on these metric values, a set of filtering methods is used to exclude biased and totally inaccurate results, in support of a more accurate calibration. Filtering methods detect imbalances introduced by OSN errors or OSN user's lack of perception (due to a widely applied survey). Two different filtering methods have been determined: *failed* and *zero score*. *Failed* filters out parameter set results, when the JSocialLib cannot retrieve data related to that specific OSN user due to authentication or authorization restrictions. This is the case when the survey participant removed permissions for the Facebook application right after the survey was submitted and, consequently, the JSocialLib methods were not able to cache any OSN data. *Zero score filtering* filters out OSN friends, who have a score value of exactly zero. If all OSN friends present a score value equalling to zero, the entire OSN user is filtered out. There are three possible cases for a zero score value: (1) the OSN user participant sabotaged the survey answers, (2) Facebook returned an error while collecting OSN data for a specific OSN friend, or (3) the lack of OSN user's perception, i.e., JSocialLib could not find any or could find only few OSN data related to the OSN user and the specified OSN friends, meaning that the OSN user participant is not self-aware of his/her own Facebook account OSN data (e.g., the OSN user thought that he/she had an interaction with friends X, Y, or Z, but actually did not send any messages, public posts, or any kind of OSN information). It is important to highlight that both *failed* and *zero score* filtering only filters out true negatives, not impacting evaluation results.

In order to determine the best parameter set for each method, all parameter set results are ordered by the *rank difference* value mean in a descending order. Therefore, iterating through each ordered parameter set, the one with the highest *top5 match count* mean value and with the lowest standard deviation is selected. With this approach, the best parameter set is selected, prioritizing the *top5 match count* mean followed by the standard deviation and rank difference mean. Table 6.11 shows the calibration outcome for both JSocialLib methods.

Table 6.11: Calibration results

Method	Parameter	Value
Interaction-based	ω_{pps}	3
	ω_{ppr}	9
	ω_{pms}	1
	ω_{pmr}	1
	t	3
Location-based	ω_{attr}	3
	ω_{cont}	7
	c_{attr}	9
	c_{cont}	7

6.2.5 Results

The JSocialLib methods are properly calibrated with the best found parameters sets, and applied to the remaining half of the OSN data obtained through the Web survey. All metrics are calculated again for the second half of the OSN data and compared to the metric values from the calibration step.

Interaction-based Method: The results based on *zero score filtering* considers 137 OSN users in the calibration and 139 OSN users in the evaluation (out of 145 OSN users each). Five OSN users were filtered out in both calibration and evaluation due to *failed filtering*. Due to *zero score filtering*, 1 and 3 OSN users were discarded, respectively, for calibration and evaluation. Moreover, in average, 1.66 and 1.88 OSN friends were removed due to *zero score filtering* from the calibration and evaluation results of each OSN user that answered the Web survey.

Figure 6.1 shows the comparison of the interaction-based method for the rank difference metric with the *failed* and *zero score filtering*. These values represented by bars indicate the rank difference mean for all OSN users ("All Results") or for each frequency group. The lower the rank difference

mean, the better the method performed. Best results were achieved for OSN users who answered "occasionally" as their interaction frequency. It is shown that the rank difference mean for the *zero score filtering* is basically the same for both calibration and evaluation data sets. For OSN users who selected "never" as an interaction frequency, evaluation results performed worse than for those calibration results. First, OSN users who answered "never" in the survey most likely do not have a good knowledge about OSN friends they have chosen to compose the ranking. Second, a low number of OSN users selected "never" as an interaction frequency (7 OSN users in total, 4 in the calibration, and 3 in the evaluation OSN data set portion). Therefore, the significance of the mean value for the "never" frequency type is lower.

Figure 6.2 shows the comparison of the interaction-based method for the *top5 match count* metric. The *top5 match count* metric shows the same values for all filtering methods, because filtering does not affect this metric. Therefore, for simplicity reasons, it only shows results comparing the calibration with the evaluation. These values indicate the *top5 match count* mean and error bars show the standard deviation. The higher *top5 match count* mean and the lower standard deviation, the better the method performed. Best results were achieved for OSN users that answered "occasionally" as their interaction frequency. Also, for the *top5 match count* metric, the interaction-based method achieves very close results for the calibration and evaluation. These measurement results for OSN users that answered "never" considerably vary due to a low confidence in the mean.

Location-based Method: The results based on zero score filtering considers 92 OSN users in the calibration and 99 OSN users in the evaluation (out of 145 OSN users each). Due to *failed filtering*, 6 calibration and 5 evaluation OSN users were discarded. Respectively for the calibration and evaluation, 40 and 39 OSN users were filtered, since they did not have any kind of location data at all and, therefore, the location-based method is not able to produce meaningful results. Moreover, 7 and 2 OSN users, respectively for the calibration and evaluation, were filtered due to *zero score filtering*. In average, 1.99 and 2.27 OSN friends were removed, respectively, from the calibration and evaluation results, due to *zero score filtering*.

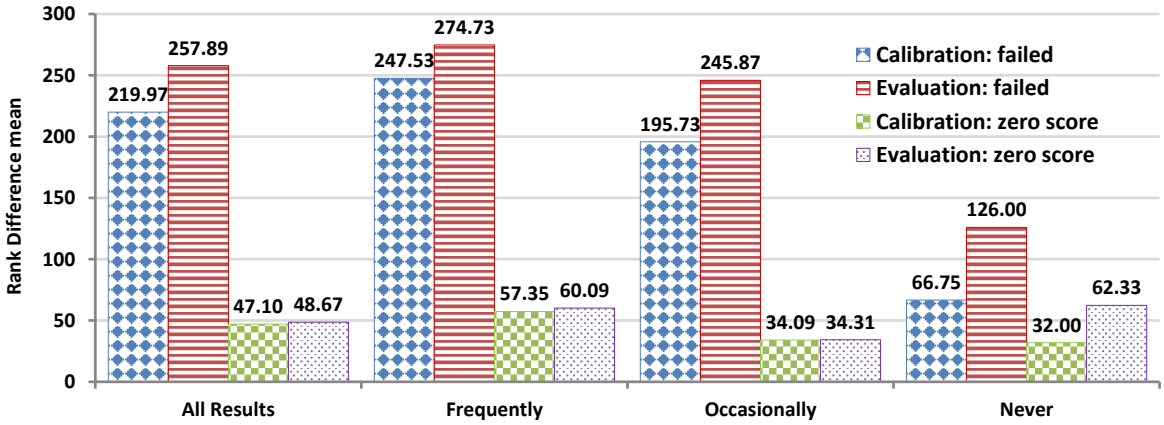

Figure 6.1: Interaction-based method: Comparison of the *rank difference* mean for the filtering methods *failed* and *zero score*

Figure 6.3 shows the comparison of the location-based method for the *rank difference* metric with the filtering methods *failed* and *zero score*. These values indicate the *rank difference* mean. For "all results", the evaluation's *rank difference* mean is higher than the calibration's *rank difference* mean. Taking into account that the *rank difference scale* is also higher compared to the interaction-based method (309.46/344.97 of location-based compared to 47.1/48.67 of interaction-based method), the evaluation of both location- and interaction-based methods had almost the same performance related to the results produced. Nevertheless, in terms of the *rank difference* metric, the location-based method performs worse when compared to the interaction-based method. An interesting observed aspect is the fact that for OSN users who answered that they "never" attach locations to their posts on Facebook the lowest *rank difference* mean is achieved – meaning that the method performs best for this particular case. In contrast to the interaction-based method, where only 4 OSN users answered with "never" as interaction frequency, 71 and 65 OSN users, respectively for the calibration and evaluation, answered that they never attach location data to their Facebook posts or messages. This result shows indicatives that the majority of OSN users and OSN friends could had been filtered out, e.g., due to the lack of attached location data (*zero score filtering*). However, the location-based method found one or more attached location data for 35 and 37 OSN users and his/her friends, respectively for calibration and evaluation. These

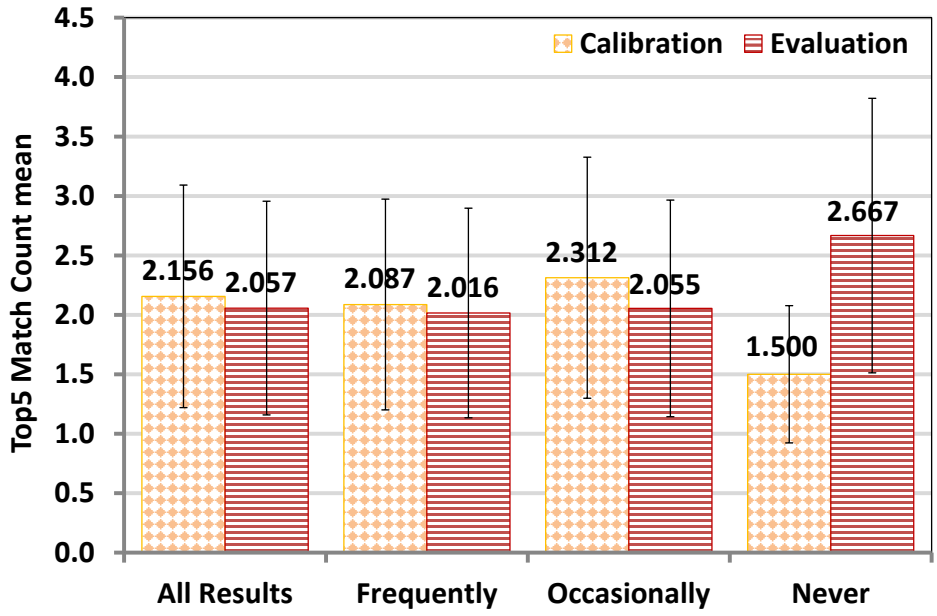

Figure 6.2: Interaction-based method: comparison of the *top5 match count*

numbers show indications that the location-based method is aligned to the OSN user perception even analyzing a minimum set of OSN data.

Figure 6.4 shows the comparison of the location-based method for the *top5 match count* metric. These values indicate the *top5 match count* mean, and the error bars represent the corresponding standard deviation. Also for the *top5 match count* metric, the interaction-based method achieves similar results for both OSN data set portions (calibration and evaluation).

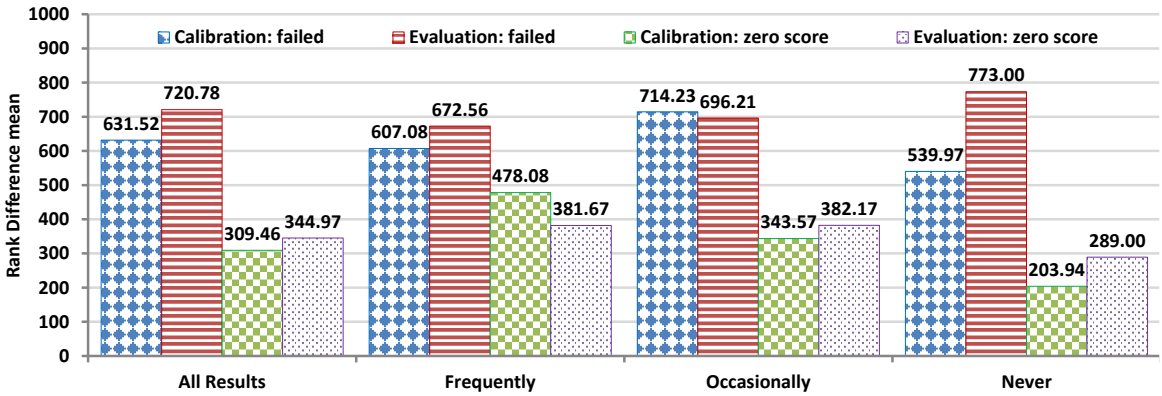

Figure 6.3: Location-based method: comparison of the *rank difference* mean for the filtering methods *failed* and *zero score*

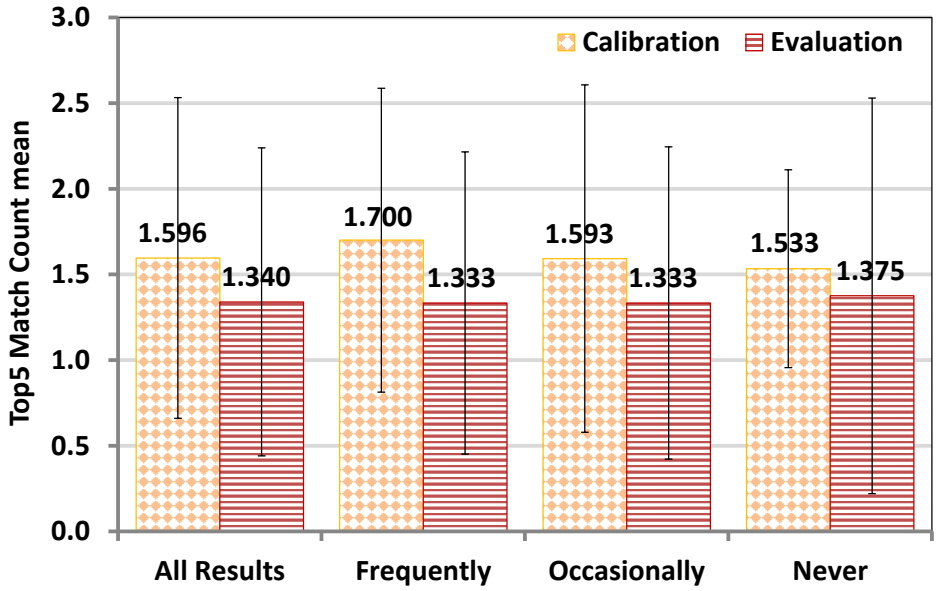

Figure 6.4: Location-based method: comparison of *top5 match count* mean

6.3 Aggregation of Heterogeneous Cloud Services' Storage

The end-to-end evaluation of the PiCsMu system, which provides the aggregation of heterogeneous Cloud services' storage, is performed by including all parts presented in Chapter 3, except the data reliability enabled by the PiCsMu FEC (Forward Error Correction). This thesis considers that the underlaying technology used by PiCsMu FEC presents a well-known overhead (defined by the Reed-Solomon code [102]) and, thus, was not included in the evaluation.

The PiCsMu system is evaluated in scenarios emulating the overall use of the system, uploading, downloading, and sharing files. Based on a set of test cases, the evaluation observes data overhead, index overhead, and the total time to upload/download files with different file sizes.

6.3.1 Set-Up

The evaluation was performed in a controlled environment [24], with 17 physical nodes interconnected to one isolated Gigabit switch. Figure 6.5 depicts the topology used and the role of each physical node. The node *n1* holds both the PiCsMu central index server and the PiCsMu IdP. The

PiCsMu P2P bootstrapping server is hosted in the node *n2*. All PiCsMu application instances are pointed to *n1* to store private indices and retrieve PiCsMu identities, and to *n2* to discover other peers in the PiCsMu P2P network. In addition, the node *n3* runs the CSG Service, which emulates a Cloud service. The CSG Service uses a software implementation – developed in the context of this thesis, only for evaluation purposes – that receives a file through HTTP and persists to a local disk, making it available for download using a specific URL pointing to the persisted file. For example, when the CSG Service receives an HTTP request to store the file *X*, it accepts the request if the file is under 5 MByte, thus, responding with an HTTP response message containing the unique URI (Unique Resource Identifier) to fetch the file. The file size restriction of 5 MByte was set by computing an empirical average of file size restrictions of public Cloud services shown in Section 6.1.2. Nodes *n1*, *n2*, and *n3* have an Intel Xeon processor, with 2.2 GHz, 48 GByte RAM, and 500 GByte 7200 RPM hard drive.

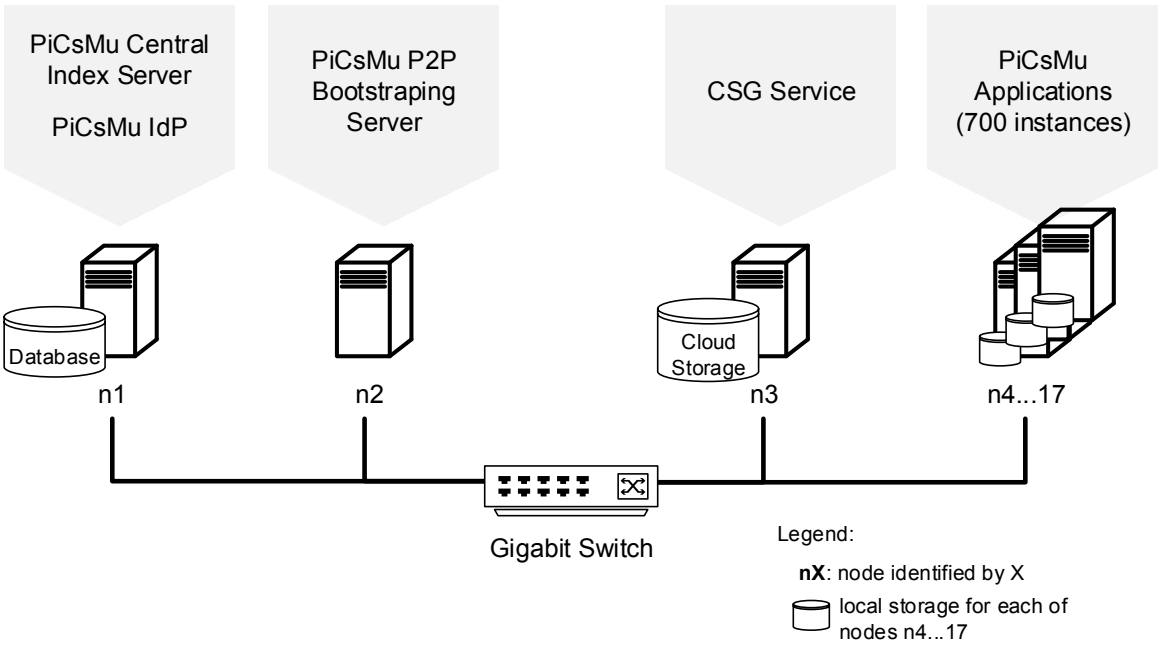

Figure 6.5: Set-up topology

Although PiCsMu has been tested and its functionality has been verified using public Cloud services, the decision to not run evaluations with all supported Cloud services' implementation relies on two reasons: first,

performing evaluations with a local service within a local network isolate the results, thus excluding, e.g., delays in public networks that could affect overall measurement values; second, the evaluation with a local service enables more test runs due to be performed in a controlled environment. The remaining 14 nodes (AMD Opteron, 24 cores, 2.5 GHz, with 64 GByte RAM, and 500 GByte 7200 RPM hard drive) were used to run PiCsMu application instances, which required about 40 MByte of RAM allowing a maximum of 700 active PiCsMu application instances (i.e., different PiCsMu users). It is important to note that the nodes' hardware configuration does not bias the overall results since the resources are shared among all PiCsMu applications, which are executed at the same time. Each of these 700 PiCsMu users follow all other remaining users in the PiCsMu social network: i.e., 1 user follows 699 PiCsMu users, and have the 699 followers. Since the evaluation aims to observe scalability, this approach ensures that the PiCsMu share notification is sent to all other user in the PiCsMu P2P network when a file is shared.

Moreover, all PiCsMu application instances are connected through a local network and listening on different ports, having all 700 PiCsMu application instances always online, thus, no churn is considered. In all experiments, all PiCsMu application instances use the same Cloud service (CSG Service) but with different Cloud services' credentials. The same files are used for each test case and test cases are repeated 10 times.

The PiCsMu scheduler implementation is configured to apply a random function when choosing a data encoder and a Cloud service credential for each file fragment. Therefore, a file is fragmented to a random number of fragments and random sizes due to the specific implementation of the chosen data encoder. However, the PiCsMu scheduler could be optimized and use a data encoder accordingly to a specific goal: e.g., prioritize to generate less fragments, prioritize to generate more fragments, prioritize to use the least amount of Cloud service's credentials as possible, among others. The evaluation chose to use a random function in order to observe the system without any optimizations being applied. Thus, the average behaviour of the system using the implemented data encoders can be taken into account.

Table 6.12: Test cases and scenarios

Scenario	Test Case A	Test Case B
1	1 PiCsMu user uploading a private file	1 PiCsMu user download a private file
2	1 PiCsMu user uploading a file to share specifically with all other PiCsMu users part of the PiCsMu IdP	All PiCsMu users downloading the shared file
3	1 PiCsMu user uploading a file to share to the whole PiCsMu P2P network	All PiCsMu users part of the P2P network downloading a file that was shared to the whole PiCsMu P2P network
4	1 User uploading a file to the CSG Service	1 User downloading a file from the CSG Service

6.3.2 Test Cases and Scenarios

The following test cases (cf. Table 6.12) were designed to cover the three modes of operation related to the file upload and download processes: (1) private storage, (2) private sharing, and (3) public sharing. Moreover, an additional test case was compiled to generate results for a comparative analysis: (4) CSGService without PiCsMu system. Test case (4) shows results of the file upload and download without the use of the PiCsMu system, which means that no additional overhead or processing time is produced. These test cases used files with sizes of 1 MByte, 10 MByte, 100 MByte, and 1 GByte in order to upload and download.

The total time to upload and download are only measured in test cases (A), since it generates the same (for scenario 1 and 4) or higher overhead/total time (for scenarios 2 and 3) as (B), representing the upper bound. Scenario 2 issues PiCsMu share notifications for all its users, resulting in a higher overhead than scenario 3, also representing the upper bound.

6.3.3 Results

The results were composed observing the following dimensions: Section 6.3.3.1 shows the file data overhead percentage (how many percent the original file grows, looking to the encoded file parts size as a result of the encoding); Section 6.3.3.2 presents the PiCsMu index overhead (how much data was generated by the PiCsMu index entities); and Section 6.3.3.3 shows the total time for file upload/download (including all file upload and download steps within the PiCsMu application). The results were obtained by parsing log files from each of the 700 PiCsMu applications, which contained the measurements related to each of these dimensions – determining the feasibility, scalability, and overhead of the PiCsMu system.

6.3.3.1 Data Overhead

Figure 6.6 shows the data overhead for test case A of scenarios (1) and (2) with a considerable data overhead variance for smaller files when compared to the 1 GByte file. Since the PiCsMu application uses a random function for the evaluation, there is a higher chance to choose different data encoders for less fragments in different runs.

Data encoders present different overhead percentage. The "best" encoder (*SteganographyEncoder*) has an overhead of 0.16% due to compressing the FFS (PNG image) after the 3 LSBs are injected, while the "worst" encoder (*ID3v2TagEncoder*) has an overhead of around 104%. The *ID3v2TagEncoder* is the one with most overhead since data is injected and represented in a hexadecimal String format within ID3v2 tags, e.g., album name, song title, song description, etc. However it is also the encoder that stores the most data (5 MByte, due to CSG Service upload size restrictions, compared to 43 KByte with *SteganographyEncoder*). For large files, the overhead levels off at around 100%, which is the average over all encoders.

Two distinct runs with extreme variations in the number of files parts were chosen to analyze the relation between data encoders and data overhead. Considering the 1 GByte file, the PiCsMu Application generated 572 encoded file parts, with a total size of 1.97 GByte data being uploaded. In

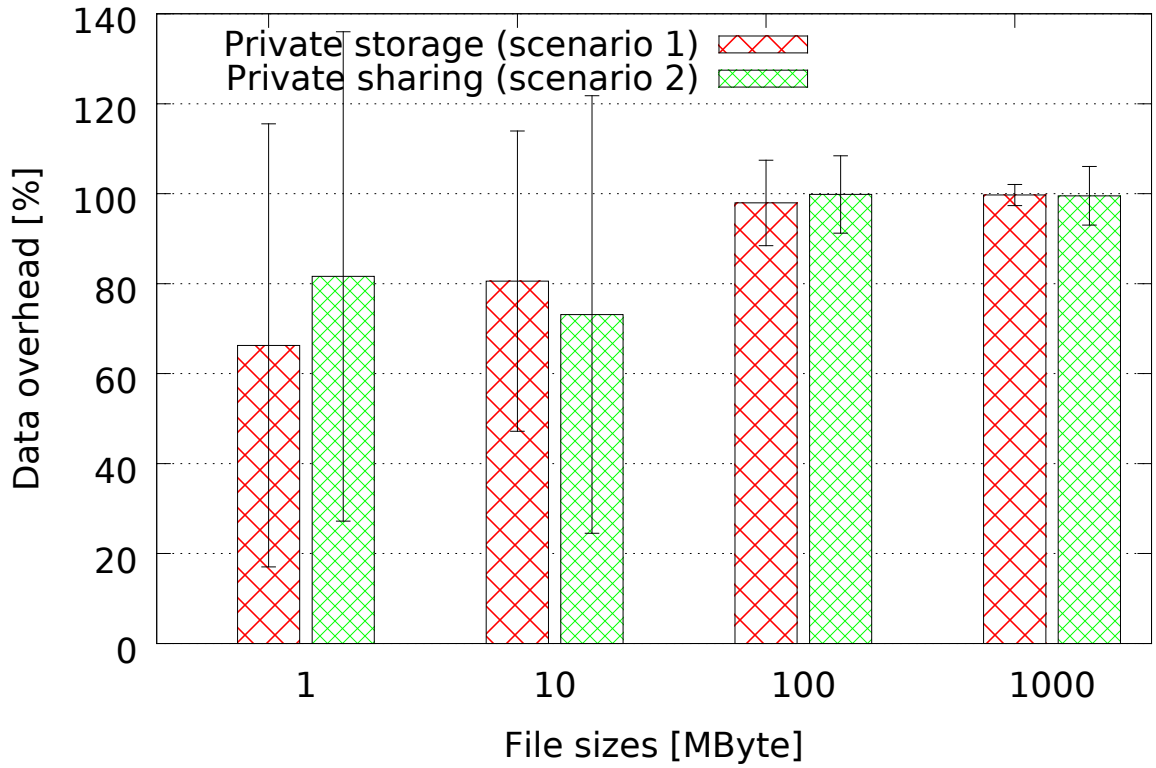

Figure 6.6: File data overhead

a second run it resulted in 237 encoded file parts, with a total size of 2.03 GByte, resulting in an additional overhead of 56 MByte.

6.3.3.2 Index Overhead

Figure 6.7 shows the index overhead for scenario 2 and 3. The index size for (2) is much smaller than for the private sharing, since the index does not need to be distributed in the DHT. For (3) the overhead to store the index in the DHT is between a factor of 2.4 to 2.7 explained by DHT's redundancy settings. It is observed that the index overhead levels off at around 2,400 Byte per peer for the private sharing, and 900 Byte for the private storage.

6.3.3.3 Total Time

Figure 6.8 shows the total time for scenarios (1), (2), and (4). In scenario (1) a 1 GByte file can be uploaded on average in 11.1 minutes (test case A) and downloaded in 6.7 minutes (test case B). When compared to the

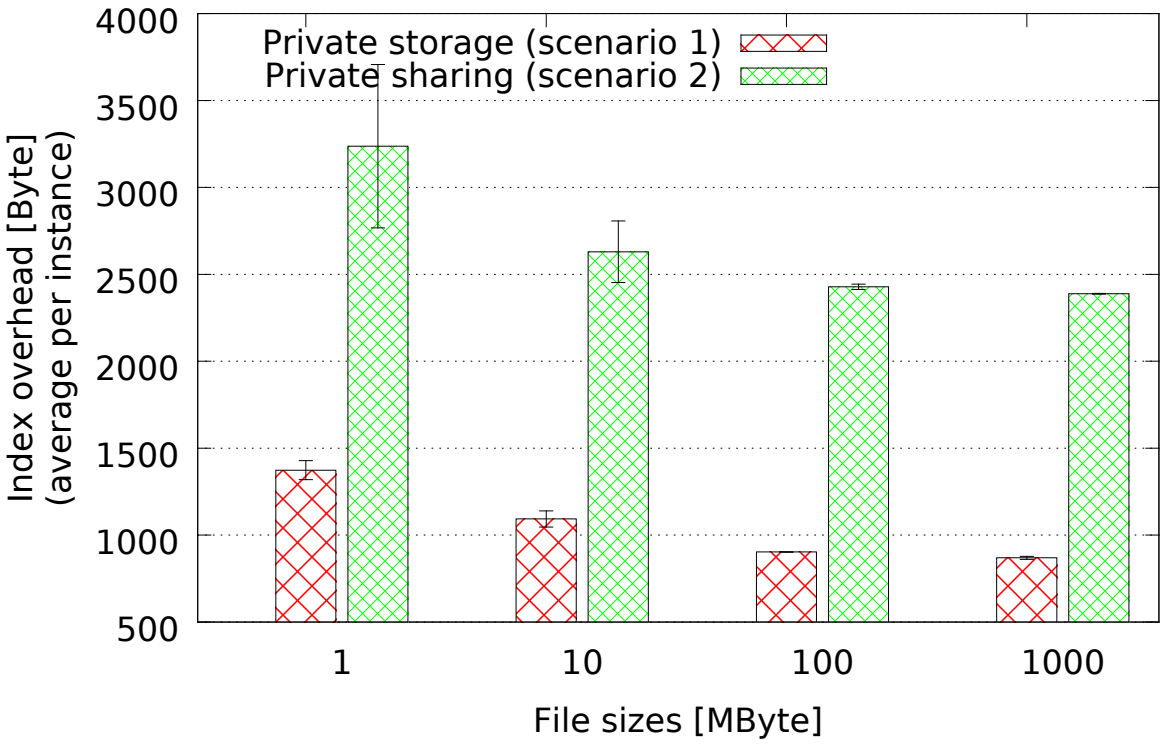

Figure 6.7: PiCsMu index overhead

average total time of scenario (4), with the same 1 GByte file, where the upload is completed in 1.59 minutes (test case A) and download is completed in 1.53 minutes (test case B), a PiCsMu user takes on average 6.9 times to upload and 4.3 times to download. This is due to fragmentation, encoding, and encryption overhead. In scenario (2), the upload and download process is faster, since the index is not encrypted and no DHT calls are performed. Another observation is that with an increasing number of file parts, the total time is increasing too, since the index for each file part is encrypted individually.

6.4 Legal Discussion

Data storage technologies in general, and especially employed by the PiCsMu system, face legal and regulative dimensions upon operation. Thus, a discussion of major related facets was prepared. First, Section 6.4.1 presents relevant technology-related aspects that allow end-users to perform data storage, in relation to Cloud storage overlays and PiCsMu. Secondly, Section 6.4.2 identifies stakeholders and points their legal role in

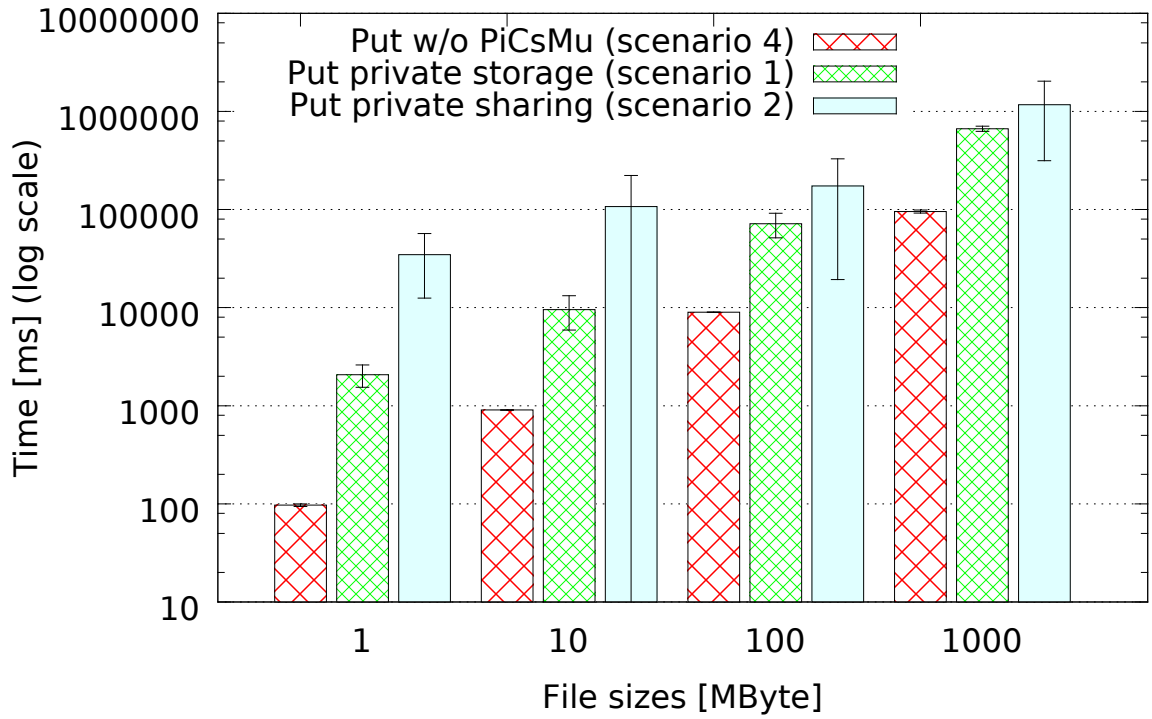

Figure 6.8: Total times for Test Case A and Scenarios 1, 2, and 4

the PiCsMu system. Third, based on technology aspects shown, the Section 6.4.3 presents three areas that are affected by regulations: representation of data in a finer level of granularity, the mechanisms of fragmentation to multiple Cloud providers, and Terms of Service from Cloud services. Lastly, Section 6.4.4 shows the status of legal decisions based on the three areas affected by regulations.

6.4.1 Relevant Technology Aspects

Data is a set of values, symbols, or variables that, coupled together, form *information* [2] – representing the lowest level of abstraction. The set of values, symbols, or variables follow a well-defined structure in which data is represented. These structures are often described in standards, since multiple entities (i.e., software) must write and read data following a data structure.

In order to perform a legal analysis about data storage using PiCsMu, it is important to note that a file (i.e., data, with a sequence of bytes) not following a certain file format is not meaningful to whom is reading/interpreting it. Both writer (i.e., who generated a file) and reader (i.e., who is read-

ing/interpreting a file) must be aware on which file format and structure the file data is represented. Therefore, if the reader can interpret the file data and generate the expected output by the writer, the data is considered *intelligible* [56]. The reader or writer can be either humans or software.

Users are able to store their files using multiple ways, including: (1) on private media, such as hard disks, USB drives, or SD cards, (2) on the Cloud, such as Dropbox [29] or Amazon [5], not being aware where exactly the files are being stored, and (3) on other users' devices, such as computers equipped with hard disks, in a Peer-to-Peer network. If users choose to share their data on cases (2) or (3), the file sharing process is made through the network and enabled by a software – available by a Cloud provider or a Peer-to-Peer (P2P) application (e.g., BitTorrent).

From the Cloud provider perspective, it is up to the provider to decide, where and how users' files are internally stored. The files can be re-organized (e.g., fragmented, encrypted) in order to suit internal restrictions, requirements, and optimization levels set by the Cloud provider. Even though files may be internally distributed or re-organized, user's files are provided in its original format when requested to download by someone. Thus, users downloading and uploading data are not aware of how files were further fragmented or where the actual data fragments are located. File fragments can be spread into the same Autonomous System from a single Cloud provider, but, at the same time, spread in different jurisdictions or politic-economic states [28]. Such process is invisible to the end-user. The majority of Cloud providers and services, such as Dropbox [29], Google Drive [48], Imgur [55], or Amazon S3 [5], employ internal data storage that are not fully transparent to end-users [52].

The same happens from a P2P application perspective, although it presents a more transparent process. Every user runs a file sharing P2P application being part of a decentralized network which every node has the same software characteristics. In other words, how a file is re-organized in order to suit the P2P application requirements is performed homogeneously by every participant of the network. Thus, if files/data should be fragmented, the process uses rules implemented in every single P2P application. It means that all other users part of this network are aware of how

the fragmentation occurred and where the fragments are located. BitTorrent is a typical example of such a process.

As an overlay system, PiCsMu adopts hybrid mechanisms in order to manage and store data, presenting different technical characteristics when legally analyzing data storage. Storage overlay applications decide how and where data is stored, using underlay services (e.g., Cloud providers and/or Peer-to-Peer networks) to actually store the data.

The particularity of storage overlay systems rely on being aware of the whole index (or metadata) information, but not being actually responsible for storing the data, i.e., keeping data with a meaningful representation under its premises. Storage overlay systems may decrease the granularity of how data is represented, thus storing a sequence of bytes that may not have a meaning for whom is storing it.

Another PiCsMu characteristic is the capability of transforming data for storage purposes. For example, PiCsMu may encode parts of a file into other well-known file formats in order to enhance security/privacy (i.e., encoding encrypted data fragments within a JPG file). Therefore, the actual file stored in a Cloud service does not characterize the original file format that the end-user aims to store.

In summary, these are the key technology aspects that are relevant for a legal discussion of Cloud storage overlays, especially for PiCsMu:

- Definition of data
- Standards for data representation
- Fragmentation of data to multiple storage facilities or entities
- Data intelligibility
- Data encoding or data transformation
- Responsibility of owning or storing only metadata information
- Responsibility of owning or storing non-intelligible data

6.4.2 Stakeholders

Considering data storage, stakeholders involved in the PiCsMu system include:

- **PiCsMu User:** PiCsMu users provide files to be privately stored, also with the possibility to share them, and perform file download, using – or not – a Cloud Service Provider. The PiCsMu user has the intelligible knowledge of the file to be stored/shared.

- **Cloud Service Provider (CSP):** CSP provides Cloud services that accept data storage in any file format (raw data), or only in specific file formats (e.g., JPG, PNG, MP3). CSPs can also sub-contract other CSPs to provide a broader range of services – e.g., Dropbox may contract Amazon S3 to add synchronization service on top of pure storage.

  CSPs employ technical mechanisms in order to:

  - Prevent Cloud Customers (CCU) to store data in CSPs' infrastructure that does not comply to the terms of service, established and accepted beforehand. The prevention occurs in the moment that CCUs specify what kind of data they want to store, employing, e.g., strong data validation mechanisms .
  - Audit its own CSP storage infrastructure anticipating possible legal issues due to data stored by CCUs, e.g., crawling CSP's infrastructure searching for files which are already stored and presents copyrighted material.

- **Cloud Customer (CCU):** Cloud Customers are customers of CSPs, having a terms of service document accepted beforehand. CCUs can also be PiCsMu users, since they might have credentials in one or multiple CSPs.

- **Regulators:** Regulators may include agencies that regulates digital contracts and Terms of Services, between CCUs and CSPs. These

agencies can, e.g., define mandatory clauses on Terms of Services and recommend how CSPs should store data of CCUs.

Other stakeholders mentioned in [118], such as, e.g., Content Provider and Cloud Operators, are not listed due to the lack of a direct contractual or technical relation to Cloud storage overlays. A Cloud storage overlay like PiCsMu is only concerned to store data in CSPs, although CSPs might have agreements to Content Providers to distribute their content. PiCsMu is also independent of Cloud Operators, since CSPs might have contracts with multiple Cloud Operators that are transparent to CCUs.

6.4.3 Regulation Areas Affected

Based on empirical observations of technical characteristics from Cloud storage overlays and its stakeholders, three questions are considered due to their legal relevance:

- Can the representation and storage of data in a finer level of granularity – thus implying to store non-intelligible data – be considered a legal problem?
- Can the process of fragmenting files (intelligible or not) to multiple Cloud providers – further manipulating such data internally – bring legal impacts in relation to data protection laws?
- What kind of legal problems Cloud storage overlays might face based on CSP's Terms of Service?

Thus, this thesis discussed the three most relevant areas potentially affected by regulations, considering the PiCsMu system in an operational manner:

- **Data Granularity:**

  [87] highlights the problematic about *intelligible* data versus distributed storage. The document is a formal response to a project proposal evaluation in UK, and published by the UK Parliament. The

following excerpt discusses the challenges of stakeholders, especially CSPs, to re-unite data from different places to constitute a intelligible data:

> *Given distributed storage and proprietary file formats, access to physical media, e.g. storage hardware in a third country, does not necessarily afford access to intelligible data. The only sure way to access intelligible data is through the user logging in to reunite fragments into intelligible form automatically. Fragments are distributed automatically; providers may or may not know in which hardware all fragments comprising one data set are stored. Some fragments may be intelligible, others not. Some providers can bypass or use customer logins, others cannot. Even providers bypassing customer logins cannot, without decryption keys, decipher data securely encrypted by controllers. Similarly, after deletion operations, fragments may or may not be intelligible or re-unitable. Again, these depend on service type and design.*

[107] discusses the legal difficulties related to data granularity in the Swedish law perspective. Although the text is dated back to 1998, the authors emphasize the emerging focus on minimal digital fragments and the problems that might bring – different ones than the traditional paper-based environment:

> *A data store-space contains "units" of data. In a traditional store one deals with units such as "books", "letters", "index cards", "formularies", and "contracts". In a comfortable, traditional situation there is a close correspondence between the unit that is asked for and retrieved and the physical object that is and can be handled as a unit. Consider, for example, a request to obtain access to the correspondence of an individual during a certain period of time. In the digital world new information handling principles begin to apply to*

> *the units of data. Fragments can be retrieved, combined, restructured, excluded, compared etc. to an extent that is simply not possible in a traditional, paper-based environment. The limits are pushed downwards so that individual micro units – a single alphabetical or numeric sign, an isolated picture element, a momentary breathing – can be identified, singled out and used. Basically, we begin to deal with patterns of ones and zeros (bit patterns) and the pattern delimiters are logical rather than physical in nature.*

As observed in [53], another issue related to data granularity is the possible ability of the provider (and of any sub-provider, e.g., IaaS provider) to access CCU's data:

> *As discussed [...], where data stored with providers are not encrypted, or only weakly encrypted, most providers have the technical ability to access the data in intelligible form. Most providers also contractually reserve the right to do so, e.g. for service/support reasons or if disclosure is compelled or requested by law enforcement authorities. If the controller/cloud customer knows that the provider has the ability and legal right to access its data, and the provider is established outside the EEA, does this mean that the controller "intended" to allow non-EEA entities to access its data? Must the controller investigate the extent of the provider's ability to access its data?*

Based on those positions expressed in different countries, the storing of non-intelligible data may raise legal problems due:

- The lack of knowledge of what someone intended to store, but allowing to store on someone's premises without any kind of inquiries;
- Privacy reasons, since non-intelligible data can be classified as personal data even not being encrypted or annonymized.

Thus, storing and disclosing non-intelligible data is considered to be sensitive.

- **Fragments in Different Locations versus Data Protection Laws:**

  The Data Protection Act (DPA) from the UK, in 1998, defines the term of "personal data" [57] as data relating to a living individual who is or can be identified either from the data or from the conjunction of data with other information (e.g., metadata). [53] discusses how CSPs and CCUs are affected by the restrictions on transferring personal data outside the European Economic Area ("EEA") under the article 25 of the Data Protection Directive 95/46/EC ("DPD") [36], termed "data export". The main point in the DPD is to protect personal data from third entities that are not within controlled jurisdictions.

  [53] discusses that restrictions on data export are not suitable with the present technology, and directives like DPD should be abolished and substituted by other more advanced and suitable controls, e.g., anonymization or encryption, and regional Clouds. Moreover, [53] states the incompatibility between current technology (enabled by Clouds) and the DPD as follows:

  > *We argue the DPD's rules should accordingly focus on restricting unauthorised access, rather than restricting data export. In other words, what matters most is not where information is stored, but who can read it, i.e. who is able to obtain access to it in intelligible form.*

  Depending on the technical set-up of CSPs' infrastructure, if authorities in a country outside European Union (EU) – where the CSP's data center is located – seize one server, or even all IT equipment of such data center, this may not necessarily mean the ability to read any personal data in intelligible form. The DPD currently only tackles the "data export" without clearly distinguishing how personal data is represented – in an intelligible or non-intelligible manner.

Another issue in this area is the manipulation and data processing of personal data in a geographically distributed manner. A Cloud storage overlay, as well as CSPs, are characterized as entities that internally further process and manipulate data if, e.g., its services/applications perform data replication, analysis, or data fragmentation. The action of "processing" personal data might also constitute legal problems based on the DPD, since the location where the data is being processed should be known and within defined geographic borders.

Thus, the fragmentation, or, in more general terms the act of processing data, may raise legal problems due to:

- The geographically distributed nature of the PiCsMu P2P network, which holds the index information of where data is stored and with which format it turns such data intelligible;
- CSPs and PiCsMu users' intention to fragment and store files in different locations, even if the data is encrypted and non-intelligible.

- **Terms of Service and Data Storage:**

  Usually, CSPs operate presenting the following clause, which is an example from Google's Terms of Service [49]:

  > *Google's Terms of Service do not allow the sending of automated queries of any sort to our system without express permission in advance from Google.*

  This clause means that the given CSP does not allow any kind of automated software to interact with its services without previously being registered to perform specific queries. Thus, Cloud storage overlays may face legal issues since they use automated means, through APIs (registered and authorized by CSPs or not), to persist data in CSPs' infrastructure.

  In addition, CSPs' terms of service often mention that the service should be used accordingly to its purposes: for example, a service

to store images, of different types (e.g., JPG, PNG), should only store personal data related to images. Therefore, CSPs should employ strong data validation rules (cf. Chapter 4) in order to:

- Verify if data pushed to CSPs infrastructure complies to the terms of service;
- Detect data transformations (e.g., data injection within accepted file formats) that could hide, e.g., illegal material.

Moreover, CSPs can also present exemption clauses in the Terms of Service that relieve them from any responsibility related to content persisted or shared by CCUs. In order to illustrate such exemption clauses the following Terms of Service's excerpt is from Dropbox [30]:

> *We may review your conduct and content for compliance with these Terms and our Acceptable Use Policy. With that said, we have no obligation to do so. We aren't responsible for the content people post and share via the Services.*

Thus, based on Terms of Service observed from CSPs, PiCsMu, and/or PiCsMu users, legal concerns are due to:

- The automated interaction with CSP's APIs;
- The intention of encoding data that are not accepted by the Terms of Services, within file formats that are accepted.

6.4.4 Status

Legal decisions regarding data storage considerably depends on the case – from technical particularities to agreed terms. Considering the areas *data granularity* and the distribution of *fragments in different locations*, one could compare the BitTorrent meta-files to the PiCsMu system, which in fact do not actually store file's data but also presents the notion of metadata represented by the "index information". Whether who holds the index or metadata violates, e.g., copyrights by linking to copyrighted material, without

the authorization of copyright holders, is highly controversial. Recently, Suprnova.org, TorrentSpy, LokiTorrent, BTJunkie, Mininova, Demonoid, Oink's Pink Palace, and, most notoriously case, The Pirate Bay, faced legal issues and trials. Most of them were shut down, with the exception of The Pirate Bay that remains online since its owners/authors are still appealing from the initial court decision.

Even though BitTorrent presents some technical similarities to PiCsMu and Cloud storage overlays, the fundamental difference is that PiCsMu primarily uses CSPs to store data, and not individuals' storage – as it happens with BitTorrent. Therefore, CSPs are responsible, as a company with legal duties within a jurisdiction, to what is being stored and to what is shared from their infrastructure. There is no legal decision, until this moment, as far as this author is aware of, that consider a Cloud storage overlay system. The known cases are only related to companies and entities that actually store data in their servers or infrastructures.

Given the fact that DPD was criticized since its creation in the EU, the General Data Protection Regulation (GDPR) [35] plans to unify data protection as a single law in Europe. Among several changes that aim to consider the advances of Cloud computing and distributed technologies, GDPR requires that any entity that performs data processing should document such processing operations – in case courts decide to access it for investigation purposes. Although GDPR is not yet approved (planned to 2016, and with a planned transition period of 2 years), the current status of such regulation shows upcoming legal issues for Cloud storage overlay systems and its users if the used technology is not adapted accordingly.

Considering the *terms of service and data storage* area, CSPs should verify and be aware of what kind of content CCUs are pushing to their servers – mentioning what are the technical mechanisms employed (e.g., describing in detail the data validation rules based on specific file's attributes). In a case involving Facebook and Power Ventures [32], which does not explicitly deal with data storage but with the use of a service, a federal judge in California state, USA, has sentenced that violating CSPs' Terms of Service does not constitute a crime. The judge also stated that "bypassing technical or code-based barriers clearly imposed by the services may violate the law

under the given jurisdiction". The use of PiCsMu and its data encoders may bring legal issues when data is stored in Cloud services where file type restrictions are mentioned in the Terms of Service. However, as a response to the case, the Electronic Frontier Foundation (EFF) argued that turning any violation of Terms of Service into a crime would empower CSPs with the decision of choosing what conduct is criminal or not [31], leaving a considerable amount of CCUs vulnerable to prosecution for everyday activities. In addition, the judge stated that it is the service provider's responsibility to declare, in the deepest level of detail, what are the "code-based barriers" in order to not misuse the provided service.

In any circumstances, courts should carefully observe the intent of each user – PiCsMu user or CCU – to base their decision upon. The malicious use intent to store illegal content using the PiCsMu system would be the starting point for legal considerations, combining previous legal decisions from different overlapping areas, such as *data granularity*, *fragment in different locations*, and *Terms of Service* abuse.

7

Summary and Conclusions

This thesis designed, implemented, and evaluated key aspects involved in allowing the aggregation of Cloud services' storage in a hybrid overlay, constituted of centralized and decentralized entities, in order to store and share data. This approach also integrated existing Online Social Networks to provide accurate measurement inputs for social recommendation, e.g., recommending new PiCsMu friends that are already friends of a PiCsMu user within existing OSNs. This thesis presented the practicability of mechanisms and components by integrating them in a Cloud storage overlay application called PiCsMu.

The PiCsMu system defines a novel storage overlay aggregating heterogeneous Cloud storage services into a single user-perceived entity, supporting both *generic* and *data-specific* storage service types in the underlay. While P2P file storage services and *generic* Cloud storage services do not restrict the data which it can be stored, *data-specific* Cloud storage services require compliance with their data restrictions, limiting the upload of files to certain file types. Thus, this thesis investigated data validation processes

of Cloud services to store arbitrary data, determining a novel problem in the area of Cloud computing. By using data encoders, data can be transformed to any format representation accepted by Cloud services' data validation process, turning it possible to store any kind of file independent of its file format. The investigation of Cloud services' data validation was carried using exploratory tests targeting several large public Cloud services, bypassing their data validation rules.

The result of file upload and download processes, which forms the PiCsMu index, allow the system to keep track on where, what, and how it stores and distributes data. The PiCsMu system deploys a decentralized architecture using entities to deal with file sharing, i.e., a DHT scheme using TomP2P, thus, avoiding centralized bottlenecks and single point of failure.

The PiCsMu system employs a set of steps to increase security and privacy: the data encoding, file part encryption, and file fragmentation processes, combined, add another layer of tasks to reconstruct original files. The combination of these steps turns it harder for an attacker to gain access to the content of original files. Moreover, the private file sharing storage mode makes use of public key encryption to securely exchange data between PiCsMu users. In this case, the data is symmetrically encrypted with a random key, which is afterwards encrypted with the receivers' public keys. This method allows fast encryption for a high numbers of receivers. PiCsMu also makes use of digital signatures in combination with the PiCsMu IdP to verify origins of content. Therefore, a PiCsMu user can always be assured that share notifications originate from declared senders.

Data reliability is designed and implemented on PiCsMu. The solution employs a scheme where multiple file fragments are stored in multiple Cloud services, preventing data loss in case a single Cloud provider shuts down its services, such as, e.g., Megaupload [130]. Moreover, the data reliability solution prevents files to be not accessible if a Cloud service does not respect an SLA in terms of data availability or time to recover. The use of Reed Solomon code allows PiCsMu to recover the original file even with multiple file fragments being lost or unavailable.

This thesis also designed, implemented, calibrated, and evaluated JSocialLib, a meta-API library providing *interaction-* and *location-based meth-*

ods in order to support the PiCsMu social recommendation system and the management of social recommendation. The interaction-based method measures the interactivity between OSN users. Thus, e.g., the PiCsMu social recommendation system is able recommend content based on what OSN friends that most interact with a certain OSN user are consuming. The location-based method measures how geographically close OSN users are to his/her OSN friends. Therefore, e.g., PiCsMu social recommendation system is able recommend new OSN friends being geographically close and also using the PiCsMu application. These methods can be used in combination to determine to whom content, items, or new friends should be recommended. Social recommendations are used to expand the user base, i.e., attract more users to PiCsMu and build its own social graph; and enhance the user's experience, e.g., recommend to share content to specific friends that might be interested, thus, producing a personalized experience.

In summary, the key contribution of this thesis is on showing the feasibility of a distributed and shareable data storage, which is capable of *aggregating storage space of heterogeneous Cloud services* to be seen as one single storage entity, relying on Cloud services' storage instead of peers' storage, being aware of data validation rules of Cloud services, and designing, implementing, and integrating a library that is able to measure interactivity and geographical closeness of friends to perform social recommendations.

7.1 Review of Contributions

This thesis is able to present key conclusions to the research questions posed:

Research Question 1.1: *Is it possible to bypass the data validation process to store arbitrary data into Cloud services' servers?*

The obtained results show that, depending on the Cloud service and data encoder applied, it is possible to store arbitrary data into the investigated Cloud services, and to bypass the data validation process.

Surprisingly, even non-sophisticated techniques as the ones implemented in FileFormatHeader- and Appender-related encoders demon-

strated to be successful when applied on Cloud services' data validation systems. In recommendation for existing Cloud providers, Cloud services should concentrate their efforts enhancing data validation algorithms to detect the use of encoders that are easier to circumvent, but can lead to a larger amount of data being stored: e.g., FileFormatHeader- and Appender-related encoders explore file format standards and are easier to circumvent compared to steganography, which requires a high complexity to detect. Even though the use of steganography can always bypass Cloud services' data validation, the amount of data injected is not as high as when using other encoders.

Research Question 1.2: *If it is possible (based on the test results), are there impacts related to security as well as accounting and charging?*

This thesis shows that there are considerable impacts in the area of security, as well as in accounting and charging. The main impact related to security is the unawareness of content persisted. This is related to security questions since Cloud services may distribute data (e.g., by sharing a picture or by turning an audio file public) that they do not filter and are not aware of. Cloud services may face legal issues due to the distribution of, e.g., illegal content under the jurisdiction where the data is persisted.

This thesis has identified accounting and charging impacts, pointing a novel dependency relation between application-specific accounting attributes (e.g., image resolution, audio length), and what the data validation process takes into consideration. In recommendation for existing Cloud providers, Cloud services should verify whether values declared in headers "reasonably" correspond to the amount of resources consumed. E.g., can a 10 second-long MP3 file, with the best possible audio quality, have 256+ MByte size, if a considerable amount of data is not present in optional headers? Even though when data validation rules follow standards (thus, being possible, e.g., to use 256 MByte just for ID3v2 tags), absurd imbalances between what is declared and what is consumed need to be taken into consideration.

Research Question 1.3: *Is it possible to build the PiCsMu system and application, exploring generic and data-specific storage of Cloud services in order to store, retrieve, and share any kind of files?*

The PiCsMu system's prototypical implementation and evaluation showed that it is possible to build an overlay with generic and data-specific Cloud services storage in the underlay. Several Cloud services are supported by PiCsMu, such as, e.g., Google Picasa, Imgur, ImageShack, Amazon S3, and SoundCloud, showing that multiple services with different file type restrictions can be used as a seamless storage unit. Moreover, the overlay implementation uses centralized (e.g., PiCsMu IdP and PiCsMu centralized index service) and decentralized entities (e.g., PiCsMu P2P network). The definition of file upload and download processes ensures the management of metadata to provide the information of what, where, and how data is stored.

Research Question 1.4: *Would the PiCsMu system scale with respect to different files (with different sizes) being stored, retrieved, and shared?*

The total time to upload/download, sharing the data privately or publicly, encapsulates all actions performed by PiCsMu to store and retrieve data – just excluding the PiCsMu Forward Error Correction. Thus, the total time measurement reveals the PiCsMu ability to handle a growing amount of data (different file sizes) in a capable manner when compared to the total time not using the PiCsMu system.

The evaluation scenario that most demands computational and network resources of the PiCsMu system is the one with a single PiCsMu user *privately sharing* a 1 GByte file for each of the 700 PiCsMu users (which are all PiCsMu users part of the PiCsMu network for evaluation). The results showed that, in this scenario, the total time to upload is, in average, 6.9 times more compared to the total time upload of a 1 GByte file *not using* PiCsMu and *not sharing* it to anyone. In the same scenario, but uploading a 1 MByte file, the total time to upload is, in average, 178.9 times more compared to the total time to upload of a 1 MByte file *not using* PiCsMu and *not sharing* it to anyone. Thus, the larger the uploaded file is, the less slow PiCsMu completes the private sharing compared to the upload with-

out using PiCsMu, nor sharing it with anyone. This proportion is observed due to (1) the parallel upload of fragments employed by PiCsMu – which does not happen when uploading a file *not using* PiCsMu; and (2) due to the recurring time to put the PiCsMu index and PiCsMu share notification entities to the 700 PiCsMu users – for all file sizes (1 MByte, 10 MByte, 100 MByte, and 1 GByte) in the *private sharing* mode.

Thus, based on the total time results collected, and observing the total time difference proportion which decreases when files are larger, this thesis concludes that PiCsMu scales with respect to different file sizes – 1 MByte, 10 MByte, 100 MByte, and 1 GByte – having 700 PiCsMu users performing upload/downloads in parallel.

Research Question 1.5: *How much data and metadata overhead is required to exploit jointly generic and data-specific storage of Cloud services considering data validation in Cloud providers?*

The PiCsMu index and share notification overhead generated when uploading and sharing a file is low. The overhead levels off at around 2,400 Byte per peer for the private sharing, and 900 Byte for the private storage, which is a non-expressive amount of data stored in the DHT and in the PiCsMu centralized index server – compared to the current capacity of personal computers' disk storage and their average speed of Internet connection [66].

The data overhead added by data encoders is considerable. For large files (100 MByte and 1 GByte) the data overhead levels off at around 100%, using a random function for choosing data encoders and splitting the file in different chunk sizes. For small files (1 MByte and 10 MByte) the data overhead has a high variation, since PiCsMu generated less file fragments: e.g., in two different runs (out of 10 runs), it presented a file overhead ratio minimum of 1.51% in one, and maximum of 115.57% in the other.

These results indicate that the PiCsMu scheduler is a key component to compose the total data and metadata overhead, as it decides how many fragments are generated. Prioritizing to generate less file parts results in less overhead, but it does not spread the data to many Cloud services, especially

not for small files. Prioritizing to generate more file parts results in more overhead, but file fragments can be spread to more Cloud services.

Thus, these results allow the conclusion that the data and metadata overhead added by the PiCsMu system is moderate.

Research Question 1.6: *How accurate can JSocialLib's interaction- and location-based methods be compared to OSN users' perception?*

The evaluation of the calibrated interaction-based method shows that it can estimate, on average, 2 out of 5 OSN friends that an OSN user also perceives as he/she interacts most with. Thus, the hypothesis presented in Section 6.2.3 was confirmed for the interaction-based method. The calibration presents that Facebook users who answered the survey perceive public interaction as more important than private interaction, i.e., PPS (Public Posts Sent) have a weight value three times higher than PMS (Private Messages Sent), while PPR (Public Posts Received) have a weight value nine times higher than PMR (Private Messages Received).

The evaluation of calibrated location-based method shows that it can estimate, on average, at least 1.3 out 5 OSN friends that an OSN user also perceives as the geographically closest to. Thus, the hypothesis presented in Section 6.2.3 was not confirmed by the location-based method. An explanation for such result is the observation that Facebook users have more interaction OSN data than location OSN data, since OSN users tend to avoid the attachment of locations to OSN content mainly due to privacy issues. Therefore, the location-based method performs worse in both metrics (*rank difference* and *top5 match count*) due to the imprecision and lack of OSN data from OSN users. These calibration results also show that OSN users consider a radius of 7 kilometers as where the geographically closest OSN friends are located, regarding users' current location.

Based on these findings, OSN users either interact more with their OSN friends with public posts than those with private messages, or they likely perceive more OSN friends who interacted publicly than privately. In addition, even with the OSN heterogeneity and OSN data collection challenges imposed, a possible implementation of the PiCsMu social recommenda-

tion system can benefit by exploring the social graph of third-party OSNs to perform recommendations aligned to OSN user's perception.

7.2 General Conclusions

This work concludes that end-users benefit from PiCsMu by having capabilities that enhance the reliability and privacy of their stored files. Instead of using and relying in one single Cloud service, e.g., Dropbox, Google Drive, SoundCloud, or Google Picasa, end-users are able to use multiple ones without being concerned about how the data is managed. At the same time that the aggregation of multiple Cloud services is beneficial for reliability purposes, it is also a key factor for user's privacy. By using PiCsMu, the files are fragmented and encrypted to multiple entities that are unable to reconstruct the original data without external support (e.g., government or court decisions). Even if encryption were not used in PiCsMu, the data persisted in one Cloud service is not intelligible, and, in turn, not presenting a meaning to be traced to the remaining fragments stored in other Cloud services.

In the area of data validation of Cloud services, this thesis concludes that encoder implementation techniques shown in the FileFormatHeader-, Steganography-, and Appender-related encoder groups can be generalized to different file formats – and not only specific ones. The implementation of different encoders depends on what is possible to explore (e.g., unused fields in file format's header) and whether storage services accept the resulting encoded files.

Moreover, this thesis concludes that the design, implementation, and use of PiCsMu system are highly important to the continuous enhancement of Cloud services. Observing the data encoders of PiCsMu, Cloud services can enhance their data validation processes to prevent the distribution of, e.g., illegal or copyrighted content, that might be stored in Cloud service's servers without their consent. However, in future, if Cloud services enhance their data validation mechanisms to detect and block non-validated data to their servers, new data encoding techniques could be plugged into

PiCsMu system. Thus, a continuous investigation following the methodology presented in this thesis (Section 4.2) should be carried.

Even if interoperability is a problem in the area of Cloud computing [80], the PiCsMu system tackles storage interoperability with the use of a Cloud storage overlay. Thus, PiCsMu turns Cloud services indirectly interoperable, since it interacts and manages different heterogeneous services to form a single and seamless storage entity.

While PiCsMu generates file parts to distribute data into multiple Cloud services, this behavior might also generate a high index volume to the PiCsMu P2P network due to index information for each file part, thus, generating more overhead. This fact shows a proportional relationship between the amount of file parts and the index size. Thus, this work concludes that, if the PiCsMu Scheduler does not use a random function to choose data encoders and Cloud services, it would need to be calibrated to not generate a large amount of file parts for large files (e.g., 1 GByte). Such calibration would optimize the index overhead, and, ultimately, maintain PiCsMu P2P network performance.

In a broader view, this work determines that the core principles developed in this thesis are also applicable to services and use cases other than storage. For example, the principle of aggregation composing an overlay can be applied to network traffic management: traffic from different routers can be aggregated in an overlay that could infer optimized network paths and dynamically adapt network circuits using virtualization. Another example is to apply the same principle to form a transparent monitoring system: aggregate multiple nodes forming an overlay of probes, where they are capable to perform active monitoring and store results in incremental fragments to the blockchain [18]. Last but not least, the methodology presented to check whether it is possible to bypass service's data validation can be used in any system that presents an input interface to users – e.g., banking systems and social media applications.

The integration of JSocialLib into applications – especially in the PiCsMu system – brings new possibilities in terms of recommendations. Therefore, the output of JSocialLib methods combined with additional in-

formation from social recommendation systems, ultimately enhance the overall user experience.

Finally, this thesis concludes that by combining centralized and decentralized entities, the PiCsMu system reaches a scalable, secure, and reliable solution to upload/download private and shared files.

7.3 Future Work

While the design, implementation, and evaluation of the PiCsMu system represent an important step into enabling the aggregation of heterogeneous Cloud services storage, also providing mechanisms to integrate existing OSNs to trigger social recommendations, open research questions still remain.

In relation to data validation in Cloud services, it is important to understand the complexity of preventing usage of data encoders classified in this thesis. Thus, solutions that detect the use of data encoders are needed by Cloud providers, focusing on constantly enhance Cloud services' data validation rules. The main question to be answered is how much computational resources are required to employ such detection solution, and optimize data validation rules over time.

Related to performance enhancements, the PiCsMu scheduler could decide the selection of Cloud services based on their current performance: the faster a Cloud service is to a specific PiCsMu user, the higher the probability of such Cloud service to store file fragments. A possible solution is to attach a distributed monitoring component to the PiCsMu application, collecting network measurements of Cloud services at a certain interval (e.g., traceroute-based measurements [79]), and sharing these results within the PiCsMu P2P network. Thus, if a PiCsMu user Y wants to share a file with PiCsMu user X, Y would first consult what is the Cloud service that X downloads data faster from – and, what is the probability of data unavailability based on history. Thus, a future design and implementation of such solution can be evaluated to quantify further gains of performance.

The PiCsMu scheduler can be also extended with the notion of rollback transactions [78]. The current version of PiCsMu does not include

data deletion if a file cannot be anymore reconstructed due to, e.g., unavailability of data, or invalid Cloud services' credentials. The PiCsMu system could observe failed uploads/downloads in certain Cloud services and delete fragments in the remaining Cloud services' credentials. Such measure will lead to save storage space and will keep the PiCsMu system always up-to-date.

Related to social recommendations, JSocialLib can be extended to perform a dynamic real-time calibration of the interactivity- and location-based methods. The calibration could be based on history and could use machine learning algorithms. An additional future direction is to implement the PiCsMu social recommendation system using the input of JSocialLib, with different social recommendations (e.g., recommendation of new content, recommendation to acquire new friends). In turn, the evaluation could investigate what is the level of acceptance of social recommendations using and not using JSocialLib methods.

In terms of a new functional addition to Cloud storage overlays, a content delivery network could be built using properties of such overlay system, focusing on file sharing. Thus, each peer of the overlay could store files that are frequently accessed by peers within or close to their Autonomous Systems [33]. The data would still be stored in Cloud services, but having a replica of frequently accessed files in the peer's local hard-disk. This approach leads to the reduction of inter-domain traffic, and traffic offload from Cloud service's datacenters. Thus, a future design and implementation of such solution would handle problems such as, e.g., detecting replicas that are not up-to-date with files stored in heterogeneous Cloud services, and electing peers that are capable to serve file's replicas within an Autonomous System.

Bibliography

[1] Charu C. Aggarwal, Joel L. Wolf, Kun-Lung Wu, and Philip S. Yu, *Horting Hatches an Egg: A New Graph-theoretic Approach to Collaborative Filtering*, 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (San Diego, California, USA), KDD 1999, ACM, 1999, pp. 201–212.

[2] Akash Mitra, DWBI Website, *Classifying Data for Successful Modeling*, 2011, `http://dwbi.org/data-modelling/dimensional-model/16-classifying-data-for-successful-modeling`, last visited: June 2016.

[3] Alexa Website, *The Web Information Company*, `http://www.alexa.com`, last visited: December 2016.

[4] Amazon.com Web Services, *Amazon S3 announces Server Side Encryption Support*, `http://aws.amazon.com/about-aws/whats-new/2011/10/04/amazon-s3-announces-server-side-encryption-support/`, last visited: October 2016.

[5] ———, *Products and Services*, `http://aws.amazon.com/products`, last visited: October 2016.

[6] Michael Ammann, *Design and Implementation of a Peer-to-Peer based System to enable the Share Functionality in a Platform-independent Cloud Storage Overlay*, Master Thesis, Department of Informatics, University of Zurich, Zurich, Switzerland, January 2013, `http://www.csg.uzh.ch/staff/machado/extern/studenttheses/Michael_Ammann_Thesis.pdf`, last visited: September 2016.

[7] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy H. Katz, Andrew Konwinski, Gunho Lee, David A. Patterson, Ariel

Rabkin, and Matei Zaharia, *Above the Clouds: A Berkeley View of Cloud Computing*, Tech. report, University of California at Berkeley, Berkeley, CA, USA, February 2009.

[8] J. Bach, *Exploratory Testing Explained*, `http://www.satisfice.com/articles/et-article.pdf`, last visited: September 2016.

[9] Marko Balabanovic and Yoav Shoham, *Fab: Content-based, collaborative recommendation*, Communications of the ACM **40** (1997), 66–72.

[10] BBC News UK Website, *Megaupload File-Sharing Site Shut Down*, `http://www.bbc.com/news/technology-16642369`, last visited: October 2016.

[11] Thomas Bocek, Ela Hunt, David Hausheer, and Burkhard Stiller, *Fast Similarity Search in Peer-to-peer Networks*, 11th IEEE/IFIP Network Operations and Management Symposium (Salvador, Brazil), NOMS 2008, April 2008.

[12] Thomas Bocek, Ela Hunt, Fabio V. Hecht, and Burkhard Stiller, *Fast Similarity Search*, `http://fastss.csg.uzh.ch`, last visited: January 2016.

[13] Thomas Bocek, Ela Hunt, and Burkhard Stiller, *Fast Similarity Search (Demo)*, `http://fastss.csg.uzh.ch`, last visited: January 2016.

[14] John S. Breese, David Heckerman, and Carl Kadie, *Empirical Analysis of Predictive Algorithms for Collaborative Filtering*, 14th Conference on Uncertainty in Artificial Intelligence (Madison, Wisconsin), UAI 1998, Morgan Kaufmann Publishers Inc., 1998, pp. 43–52.

[15] David Brumley, Pongsin Poosankam, Dawn Song, and Jiang Zheng, *Automatic Patch-Based Exploit Generation is Possible: Techniques and Implications*, 2008 IEEE Symposium on Security and Privacy (Washington, DC, USA), SP 2008, IEEE Computer Society, 2008, pp. 143–157.

[16] Business Insider Website, *Amazon's Cloud Crash Disaster Permanently Destroyed Many Customers' Data*, `http://www.businessinsider.com/amazon-lost-data-2011-4`, last visited: October 2016.

[17] Li Chao, Yu Jian, Li Xiang, and Chen Jia Hui, *A Social Network System Oriented Hybrid Recommendation Model*, 2nd International Conference on Computer Science and Network Technology (Changchun, China), ICCSNT 2012, 2012, pp. 901–906.

[18] K. Christidis and M. Devetsikiotis, *Blockchains and Smart Contracts for the Internet of Things*, IEEE Access **4** (2016), 2292–2303.

[19] Cheng-Hao Chu, Wan-Chuen Wu, Cheng-Chi Wang, Tzung-Shi Chen, and Jen-Jee Chen, *Friend Recommendation for Location-Based Mobile Social Networks*, 7th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (Taichung, Taiwan), IMIS 2013, 2013, pp. 365–370.

[20] Simone Cirani and Luca Veltri, *A Multicast-Based Bootstrap Mechanism for Self-Organizing P2P Networks*, IEEE Global Telecommunications Conference, GLOBECOM 2009, December 2009, pp. 1–6.

[21] Ian Clarke, Oskar Sandberg, Matthew Toseland, and Vilhelm Verendel, *Private Communication Through a Network of Trusted Connections: The Dark Freenet*, 2010, `https://freenetproject.org/papers.html`, last visited: October 2016.

[22] CloudHQ Website, *Salesforce, Box, Google Drive, Gmail, Dropbox, SharePoint, OneDrive, Evernote, and Basecamp - sync and integrated by cloudHQ*, `https://www.cloudhq.net/`, last visited: October 2016.

[23] CloudKafé Website, *Organizing Your Cloud*, `https://www.cloudkafe.com/`, last visited: October 2016.

[24] Communication Systems Group (CSG), *Testbed Infrastructure for Research Activities*, `http://www.csg.uzh.ch/services/testbed/`, last visited: May 2016.

[25] Ruben Cuevas, Roberto Gonzalez, Angel Cuevas, and Carmen Guerrero, *Understanding the Locality Effect in Twitter: Measurement and Analysis*, Personal and Ubiquitous Computing **18** (2014), no. 2, 397–411.

[26] F.C. Delicato, P.F. Pires, L. Pirmez, and T. Batista, *Wireless Sensor Networks as a Service*, 17th IEEE InternationalConference and Workshops on Engineering of Computer Based Systems (Oxford, UK), ECBS 2010, 2010, pp. 410–417.

[27] Mukund Deshpande and George Karypis, *Item-based top-N Recommendation Algorithms*, ACM Trans. Inf. Syst. **22** (2004), no. 1, 143–177.

[28] Daniel Dönni, Guilherme Sperb Machado, Christos Tsiaras, and Burkhard Stiller, *Schengen Routing: A Compliance Analysis*, Intelligent Mechanisms for Network Configuration and Security (Ghent, Belgium), AIMS 2015,

Springer, June 2015, pp. 100–112.

[29] Dropbox Website, *Dropbox Storage Service*, http://www.dropbox.com, last visited: October 2016.

[30] ———, *Terms of Service*, https://www.dropbox.com/terms, last visited: February 2016.

[31] Electronic Frontier Foundation, *Brief of Amicus Curiae Eletronic Frontier Foundation in Support of Defendant Power Ventures' Motion For Summary Judgement on Cal. Penal Code 502(c)*, July 2010, https://www.eff.org/files/filenode/facebook_v_power/fbvpower_june_amicus_final.pdf, last visited: July 2016.

[32] Electronic Frontier Foundation, Marcia Hofmann, *Court: Violating Terms of Service Is Not a Crime, But Bypassing Technical Barriers Might Be*, July 2010, https://www.eff.org/deeplinks/2010/07/court-violating-terms-service-not-crime-bypassing, last visited: July 2016.

[33] Michael Enz, *Supporting Data Distribution Optimization in a Network of Heterogeneous Storage Services based on Distributed Measurements of Network Monitoring Metrics*, Master Thesis, Department of Informatics, University of Zurich, Zurich, Switzerland, August 2014, https://files.ifi.uzh.ch/CSG/staff/machado/extern/studenttheses/Michael_Enz_MA_Thesis.pdf, last visited: September 2016.

[34] Patrick Th. Eugster, Pascal A. Felber, Rachid Guerraoui, and Anne-Marie Kermarrec, *The Many Faces of Publish/Subscribe*, ACM Computing Surveys **35** (2003), no. 2, 114–131.

[35] European Commission, *Proposal for the EU General Data Protection Regulation*, January 2012, http://ec.europa.eu/justice/data-protection/document/review2012/com_2012_11_en.pdf, last visited: June 2016.

[36] European Parliament and the Council of European Union, *Directive 95/46/EC of the European Parliament and the Council – Part 2*, October 1995, http://ec.europa.eu/justice/policies/privacy/docs/95-46-ce/dir1995-46_part2_en.pdf, last visited: July 2016.

[37] Facebook Developer Website, *API Rate Limit*, https://developers.facebook.com/docs/reference/ads-api/api-rate-limiting/, last visited: January 2016.

[38] ———, *Bug Report at November 25th, 2013*, https://developers.facebook.com/x/bugs/694034083948939, last visited: October 2016.

[39] Facebook Website, *Facebook*, https://www.facebook.com, last visited: October 2016.

[40] Alexander Filitz, *Online Social Network Capabilities for a Platform-Independent Cloud Storage System for Multi-Usage*, Master Thesis, Department of Informatics, University of Zurich, Zurich, Switzerland, January 2014, http://publication.pics.mu/thesis/Alexander_Filitz_MA_thesis.pdf, last visited: September 2016.

[41] J. Fridrich, M. Goljan, and R. Du, *Steganalysis based on JPEG Compatibility*, Proceedings of Special Session on Theoretical and Practical Issues in Digital Watermarking and Data Hiding, SPIE 2001, vol. 4518, 2001, pp. 275–280.

[42] Jan Gabrielsson, Ola Hubertsson, Ignacio Más, and Robert Skog, *Cloud Computing in Telecommunications*, Tech. report, Ericsson, 2010.

[43] V. Gancheva, *Data Security and Validation Framework for a Scientific Data Processing SOA Based System*, Developments in E-systems Engineering, 2011, DeSE 2011, December 2011, pp. 576–580.

[44] W. Gellert, H. Kustner, M. Hellwich, and H.; (eds.) Kaestner, *The VNR Concise Encyclopedia of Mathematics*, Van Nostrand Reinhold Company, 1975.

[45] Frank Gillet, *Conventional wisdom is wrong about IaaS*, Tech. report, Forrester Research, 2009.

[46] M. Gjoka, M. Kurant, C.T. Butts, and A. Markopoulou, *Walking in Facebook: A Case Study of Unbiased Sampling of OSNs*, IEEE INFOCOM (San Diego, CA, USA), 2010, pp. 1–9.

[47] Google+ API Client Website, *A Java client for the Google+ API*, https://github.com/Glamdring/google-plus-java-api/, last visited: February 2016.

[48] Google Corporate Website, *Google Products and Solutions*, http://google.com/intl/en/about/products/, last visited:

October 2016.

[49] Google Support Website, *Search Console Help: Automated Queries*, https://support.google.com/webmasters/answer/66357, last visited: July 2016.

[50] D. Groenewegen and E. Visser, *Integration of Data Validation and User Interface Concerns in a DSL for Web Applications*, Software & Systems Modeling, vol. 12, Springer-Verlag, 2013, pp. 35–52.

[51] Harvard Library Website, *Harvard Library Portal*, http://http://library.harvard.edu, last visited: July 2016.

[52] W. Kuan Hon and Christopher Millard, *Data Export in Cloud Computing – How Can Personal Data Be Transferred Outside the EEA? The Cloud of Unknowing, Part 4*, Script-ed **9** (2012), no. 25, 29–63.

[53] W. Kuan Hon, Christopher Millard, and Ian Walden, *Who is Responsible for Personal Data in Cloud Computing? – The Cloud of Unknowing, Part 2*, International Data Privacy Law **2** (2012), no. 1, 3–18.

[54] ImageShack Website, *Best Place for All of Your Image Hosting and Image Sharing Needs*, https://www.imageshack.com, last visited: October 2016.

[55] Imgur Website, *The Simple Image Sharer*, https://www.imgur.com, last visited: October 2016.

[56] Information Commissioner's Office, *Guide to Data Protection: Principle 6 – Rights, Subject Access Request*, https://ico.org.uk/for-organisations/guide-to-data-protection/principle-6-rights/subject-access-request/, last visited: July 2016.

[57] ———, *What is personal data? – A quick reference guide*, 2012, https://ico.org.uk/media/for-organisations/documents/1549/determining_what_is_personal_data_quick_reference_guide.pdf, last visited: June 2016.

[58] Instagram Website, *Capture and Share the World's Moments*, https://www.instagram.com, last visited: October 2016.

[59] N. F. Johnson and S. Jajodia, *Steganalysis of Images Created Using Current Steganography Software*, Information Hiding, Lecture Notes in Computer Science, vol. 1525, Springer Berlin Heidelberg, 1998, pp. 273–289.

[60] JoliCLoud Website, *The new way to manage your storage*, https://www.jolicloud.com/, last visited: October 2016.

[61] JSON-RPC Website Specification, *A Light Weight Remote Procedure Call Protocol, Specification 2.0*, http://jsonrpc.org, last visited: February 2016.

[62] JSTOR Website, *Journal Storage*, http://www.jstor.org, last visited: July 2016.

[63] Andreas Kaltenbrunner, Salvatore Scellato, Yana Volkovich, David Laniado, Dave Currie, Erik J. Jutemar, and Cecilia Mascolo, *Far From the Eyes, Close on the Web: Impact of Geographic Distance on Online Social Interactions*, ACM workshop on Workshop on Online Social Networks (Helsinki, Finland), WOSN 2012, ACM, July 2012, pp. 19–24.

[64] G. C. Kessler, *An Overview of Steganography for the Computer Forensics Examiner*, February 2014, http://www.garykessler.net/library/fsc_stego.html., last visited: October 2016.

[65] Jong Kim and Young Kyun Park, *Virtual File System Integrating Multiple Cloud Storage Services and Operating Method of the Same*, December 2013, http://www.google.com/patents/US20140164449, last visited: July 2016.

[66] KPCB – Kleiner Perkins Caufield & Byers, *Internet Trends 2015*, May 2015, http://www.kpcb.com/file/kpcb-internet-trends-2015, last visited: June 2016.

[67] Balachander Krishnamurthy, Phillipa Gill, and Martin Arlitt, *A Few Chirps About Twitter*, Proceedings of the First Workshop on Online Social Networks (Seattle, WA, USA), WOSN 2008, 2008, pp. 19–24.

[68] P. Leach, M. Mealling, and R. Salz, *A Universally Unique IDentifier (UUID) URN Namespace*, RFC 4122, July 2005.

[69] Alexander Lenk, Markus Klems, Jens Nimis, Stefan Tai, and Thomas Sandholm, *What's inside the Cloud? An architectural map of the Cloud landscape*, 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing (Washington, DC, USA), CLOUD 2009, IEEE Computer Society, 2009, pp. 23–31.

[70] V. I. Levenshtein, *Binary Codes Capable of Correcting Deletions, Insertions and Reversals*, Soviet Physics Doklady **10** (1966), no. 8, 707–710.

[71] David Liben-Nowell, Jasmine Novak, Ravi Kumar, Prabhakar Raghavan, and Andrew Tomkins, *Geographic Routing in Social Networks*, Proceedings of the National Academy of Sciences of the United States of America **102** (2005), no. 33, 11623–11628.

[72] Eng Keong Lua, Jon Crowcroft, Marcelo Pias, Ravi Sharma, and Steven Lim, *A Survey and Comparison of Peer-to-Peer Overlay Network Schemes*, IEEE Communications Surveys and Tutorials **7** (2005), 72–93.

[73] Guilherme Sperb Machado, Thomas Bocek, Michael Ammann, and Burkhard Stiller, *A Cloud Storage Overlay to Aggregate Heterogeneous Cloud Services*, Technical Report No. 2013.0005, Institute of Informatics, University of Zurich, Zürich, Switzerland, May 2013, `https://files.ifi.uzh.ch/CSG/staff/machado/extern/publications/IFI-2013.0005.pdf`, last visited: September 2016.

[74] ———, *A Cloud Storage overlay to aggregate heterogeneous Cloud services*, 38th IEEE Conference on Local Computer Networks, LCN 2013, October 2013, pp. 597–605.

[75] Guilherme Sperb Machado, Thomas Bocek, Alexander Filitz, and Burkhard Stiller, *Design, Prototyping, and Evaluation of Methods to Measure Interactivity and Geographical Closeness of Online Social Network Users in Support of Social Recommendation Systems*, Technical Report No. 2014.0003, Institute of Informatics, University of Zurich, Zürich, Switzerland, May 2014, `http://publication.pics.mu/papers/IFI-2014.0003.pdf`, last visited: September 2016.

[76] ———, *Measuring Interactivity and Geographical Closeness of Online Social Network users to Support Social Recommendation Systems*, 10th International Conference on Network and Service Management (Rio de Janeiro, Brazil), CNSM 2014, November 2014, pp. 187–192.

[77] Guilherme Sperb Machado, Thomas Bocek, and Burkhard Stiller, *PiCsMu: A System to Aggregate Multiple Heterogeneous Cloud Services' Storage*, IFIP/IEEE Network Operations and Management Symposium, NOMS 2014, May 2014, pp. 1–2.

[78] Guilherme Sperb Machado, Fabio Daitx, Weverton Cordeiro, Cristiano Both, Luciano Gaspary, Lisandro Granville, Claudio Bartolini, Akhil Sahai, David Trastour, and Katia Saikoski, *Enabling Rollback Support in IT Change Management Systems*, IFIP/IEEE Network Operations and Management

Symposium (Salvador, Brazil), NOMS 2008, April 2008, pp. 347–354.

[79] Guilherme Sperb Machado, Michael Enz, and Burkhard Stiller, *TraceMan: A Traceroute-based Measurement and Management Tool*, Praxis der Informationsverarbeitung und Kommunikation **36** (2013), no. 1, 52.

[80] Guilherme Sperb Machado, David Hausheer, and Burkhard Stiller, *Considerations on the Interoperability of and between Cloud Computing Standards*, G2CNet Workshop: From Grid to Cloud Networks (Banff, Canada), OGF 27, October 2009, pp. 1–4.

[81] Guilherme Sperb Machado, Fabio Hecht, Martin Waldburger, and Burkhard Stiller, *Bypassing Cloud Providers' Data Validation to Store Arbitrary Data*, IFIP/IEEE International Symposium on Integrated Network Management, IM 2013, May 2013, pp. 1–8.

[82] Guilherme Sperb Machado and Burkhard Stiller, *Investigations of an SLA Support System for Cloud Computing (SLACC)*, Praxis der Informationsverarbeitung und Kommunikation **34** (2011), no. 2, 80–86.

[83] J. G. McNeff, *The Global Positioning System*, IEEE Transactions on Microwave Theory and Techniques **50** (2002), no. 3, 645–652.

[84] Peter Mell and Timothy Grance, *The NIST Definition of Cloud Computing*, Recommendations of the National Institute of Standards and Technology (NIST), `http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf`, last visited: October 2016, 2011.

[85] Shicong Meng and Ling Liu, *Enhanced Monitoring-as-a-Service for Effective Cloud Management*, IEEE Transactions on Computers **62** (2013), no. 9, 1705–1720.

[86] Microsoft Website, *SkyDrive, Files Anywhere*, `http://www.skydrive.com`, last visited: October 2016.

[87] Christopher Millard, Alan Cunningham, and Kuan Hon, *UK Parliament, Justice Committee: Written evidence from Christopher Millard, Alan Cunningham and Kuan Hon, Cloud Legal Project, 2012*, `http://www.publications.parliament.uk/pa/cm201213/cmselect/cmjust/572/572vw09.htm`, last visited: October 2016.

[88] Dejan Milojicic, *Cloud Computing: Interview with Russ Daniels and Franco Travostino*, IEEE Internet Computing **12** (2008), no. 5, 7–9.

[89] MultCloud website, *Manage, Move, Copy, and Migrate Files Between Cloud Storage Services with MultCloud*, https://www.multcloud.com/, last visited: October 2016.

[90] J. Naruchitparames, M.H. Gunes, and S.J. Louis, *Friend Recommendations in Social Networks using Genetic Algorithms and Network Topology*, IEEE Congress on Evolutionary Computation (New Orleans, LA, USA), EC 2011, 2011, pp. 2207–2214.

[91] National Institute of Standards and Technology, *Secure Hash Standard (SHS)*, U.S. Dept. of Commerce, National Institute of Standards and Technology (NIST), Gaithersburg, MD, USA, 2008.

[92] Roozbeh Nia, Fredrik Erlandsson, Henric Johnson, and S. Felix Wu, *Leveraging Social Interactions to Suggest Friends*, 33rd IEEE International Conference on Distributed Computing Systems Workshops (Philadelphia, PA, USA), ICDCSW 2013, 2013, pp. 386–391.

[93] M. Nilsson, *ID3 Tag Version 2.3 – Informal Standard*, February 1999, http://www.id3.org/id3v2.3.0., last visited: December 2016.

[94] NXDrive Website, *Secure data storage with privacy protection*, https://www.nxdrive.com/, last visited: October 2016.

[95] OAuth Specification, *OAuth 2.0 Specification*, http://oauth.net/2, last visited: October 2016.

[96] Goldreich Oded, *Foundations of Cryptography: Volume 2, Basic Applications*, 1st ed., Cambridge University Press, New York, NY, USA, 2009.

[97] OpenSocial Website, *OpenSocial Project Standards*, http://opensocial.org, last visited: October 2016.

[98] Otixo, *All Your Cloud Files from a Single Login*, http://www.otixo.com, last visited: October 2016.

[99] Sanjoy Paul, Shubhashis Sengupta, Annervaz Karukapadath Mohamedrasheed, Amitabh Saxena, and Vikrant Kaulgud, *Secure Online Distributed Data Storage Services*, January 2014, http://www.google.com/patents/US20140201541, last visited: July 2016.

[100] PCWorld Website, *Hotmail Data Loss Reveals Cloud Trust Issues*, http://www.pcworld.com/article/215365/hotmail_data_loss_reveals_cloud_trust_issues.html, last visited: October 2016.

[101] A. Petukhov and D. Kozlov, *Detecting Security Vulnerabilities in Web Applications Using Dynamic Analysis with Penetration Testing*, Application Security Conference (Ghent, Belgium), AppSec 2008, May 2008.

[102] James S. Plank, *A Tutorial on Reed-Solomon Coding for Fault-Tolerance in RAID-like Systems*, vol. 27, John Wiley & Sons, Inc., 1997, pp. 995–1012.

[103] V. Podobnik, D. Striga, A. Jandras, and I. Lovrek, *How to calculate Trust Between Social Network Users?*, 20th International Conference on Software, Telecommunications, and Computer Networks (Split, Croatia), SoftCOM 2012, 2012, pp. 1–6.

[104] RestFB Library Website, *A Lightweight Java Facebook Graph API and Old REST API Client*, `http://www.restfb.com`, last visited: October 2016.

[105] R. Rivest, *The MD5 Message-Digest Algorithm*, RFC 1321, April 1992, Updated by RFC 6151.

[106] C. S. Sapp, *WAVE PCM SoundFile Format: Project for the lecture Perceptual Audio Coding*, August 2012, `https://ccrma.stanford.edu/courses/422/projects/WaveFormat/`, last visited: December 2016.

[107] Peter Seipel, *Borderline Technology: Its Difficulties in a Swedish Perspective*, September 1998, `http://www.juridicum.su.se/iri/docs/Borderline_Technology/`, last visited: July 2016.

[108] Amit Singhal, *Modern Information Retrieval: A Brief Overview*, IEEE Data Engineering Bulletin **24** (2001), no. 4, 35–43.

[109] SoundCloud Website, *Become a SoundCloud Pro*, `https://soundcloud.com/pro`, last visited: July 2016.

[110] ———, *Hear the World's Sounds*, `http://www.soundcloud.com`, last visited: October 2016.

[111] SourceForge Website, *JHOVE Project – File validation and characterization*, `http://sourceforge.net/projects/jhove/`, last visited: July 2016.

[112] SpiderOak Website, *Private Online Backup and Sharing*, `https://www.spideroak.com`, last visited: October 2016.

[113] ITU Standard, *Digital Compression and Coding of Continuous-Tone Still Images*, September 1992, `http://www.w3.org/Graphics/JPEG/itu-t81.pdf`, last visited: December 2016.

[114] Ralf Steinmetz and Klaus Wehrle, *Peer-to-Peer Systems and Applications (Lecture Notes in Computer Science)*, Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[115] Daniel Stutzbach and Reza Rejaie, *Understanding Churn in Peer-to-Peer Networks*, Proceedings of the 6th ACM SIGCOMM conference on Internet measurement (Rio de Janeriro, Brazil), IMC 2006, ACM, 2006, pp. 189–202.

[116] E. Swanson, C. J. Ganier, R. Holman, and J. Rosser, *Steganography in Signals: Lecture Project*, August 2001, `http://www.clear.rice.edu/elec301/Projects01/smokey_steg/.`, last visited: October 2016.

[117] H. Takabi and J.B.D. Joshi, *Policy Management as a Service: An Approach to Manage Policy Heterogeneity in Cloud Computing Environment*, 45th Hawaii International Conference on System Science (Hawaii, USA), HICSS 2012, 2012, pp. 5500–5508.

[118] Telecommunication Standardization Sector of ITU, *ITU-T Recommendation Y.3013: Socio-economic Assessment of Future Networks by Tussle Analysis*, August 2014, `https://www.itu.int/rec/dologin_pub.asp?lang=e&id=T-REC-Y.3013-201408-I!!PDF-E&type=items`, last visited: July 2016.

[119] TomP2P Website, *A P2P-based High Performance Key-value Pair Storage Library*, `https://www.tomp2p.net`, last visited: October 2016.

[120] TwitPic Website, *Share your Photos and Videos on Twitter*, `https://www.twitpic.com`, last visited: October 2016.

[121] Twitter4J Library Website, *A Java Library for the Twitter API*, `http://twitter4j.org`, last visited: October 2016.

[122] Ubuntu Website, *MAAS: Metal-as-a-Service*, `http://maas.ubuntu.com`, last visited: September 2016.

[123] Guido Urdaneta, Guillaume Pierre, and Maarten Van Steen, *A Survey of DHT Security Techniques*, ACM Comput. Surv. **43** (2011), 8:1–8:49.

[124] Masoud Valafar, Reza Rejaie, and Walter Willinger, *Beyond Friendship Graphs: A Study of User Interactions in Flickr*, Proceedings of the 2Nd ACM Workshop on Online Social Networks (Barcelona, Spain), WOSN 2009, 2009, pp. 25–30.

[125] Toby Velte, Anthony Velte, and Robert Elsenpeter, *Cloud Computing, A Practical Approach*, 1 ed., McGraw-Hill, Inc., New York, NY, USA, 2010.

[126] Cong Wang, Qian Wang, Kui Ren, Ning Cao, and Wenjing Lou, *Toward Secure and Dependable Storage Services in Cloud Computing*, IEEE Transactions on Services Computing **5** (2012), no. 2, 220–232.

[127] BBC News UK Website, *Big Data: Are You Ready for Blast-Off?*, `http://www.bbc.com/news/business-26383058`, last visited: October 2016.

[128] W3C Website, *Portable Network Graphics (PNG) Specification*, November 2003, `http://www.w3.org/TR/PNG/`, last visited: December 2016.

[129] Wikipedia Website, *GMailFS*, `http://en.wikipedia.org/wiki/GMailFS`, last visited: October 2016.

[130] ———, *Megaupload*, `http://en.wikipedia.org/wiki/Megaupload`, last visited: October 2016.

[131] Christo Wilson, Bryce Boe, Alessandra Sala, Krishna P.N. Puttaswamy, and Ben Y. Zhao, *User Interactions in Social Networks and Their Implications*, 4th ACM European Conference on Computer Systems (Nuremberg, Germany), EuroSys 2009, 2009, pp. 205–218.

[132] David Issac Wolinsky, Pierre St. Juste, P. Oscar Boykin, and Renato Figueiredo, *Addressing the P2P Bootstrap Problem for Small Overlay Networks*, IEEE 10th International Conference on Peer-to-Peer Computing, P2P 2010, August 2010, pp. 1–10.

[133] Wuala Website, *Secure Storage Service*, `http://www.wuala.com`, last visited: October 2016.

[134] Changchun Yang, Jing Sun, and Ziyi Zhao, *Personalized Recommendation based on Collaborative Filtering in Social Network*, IEEE International Conference on Progress in Informatics and Computing (Shanghai, China), PIC 2010, vol. 1, 2010, pp. 670–673.

[135] J.P. Yoon, *Techniques for Data and Rule Validation in Knowledge Based Systems*, 4th Conference on Systems Integrity, Software Safety and Process Security, COMPASS 1989, June 1989, pp. 62–70.

[136] Lian Yu, Wei-Tek Tsai, Xiangji Chen, Linqing Liu, Yan Zhao, Liangjie Tang, and Wei Zhao, *Testing as a Service over Cloud*, 5th IEEE International Symposium on Service Oriented System Engineering (Nanjing, China), SOSE 2010, IEEE Computer Society, 2010, pp. 181–188.

[137] Xiao Yu, Ang Pan, Lu-An Tang, Zhenhui Li, and Jiawei Han, *Geo-Friends Recommendation in GPS-based Cyber-physical Social Network*, International Conference on Advances in Social Networks Analysis and Mining (Kaohsiung, Taiwan), ASONAM 2011, 2011, pp. 361–368.

[138] Jing Zhang, Jie Tang, Bangyong Liang, Zi Yang, Sijie Wang, Jingjing Zuo, and Juanzi Li, *Recommendation over a Heterogeneous Social Network*, 9th International Conference pn Web-Age Information Management, WAIM 2008, 2008, pp. 309–316.

Appendix

A.1 Reed-Solomon Code: Encoding and Decoding

This appendix displays the Reed-Solomon code, which is the scheme used by the PiCsMu System to implement Forward Error Correction (FEC).

Reed-Solomon can correct errors and erasures [102]. An error is defined when bits or bytes in a data symbol does not reflect what was initially transmitted and their position is not known. An erasure is defined when the position of a data symbol which contains errors is known at the moment of receiving the data.

The Reed-Solomon algorithm is specified as RS(n,k), with s-bits data symbols. The Reed-Solomon encoding process takes k data symbols of s bits and generates parity data symbols which forms a sequence of n data symbols, called a codeword. Within such generated codeword of n data symbols, there are p parity symbols of s bytes each, where $p = n - k$. The maximum value of n (codeword size) for a Reed-Solomon code is $n = 2^s - 1$. Figure A.1 illustrates the RS(n,k) code.

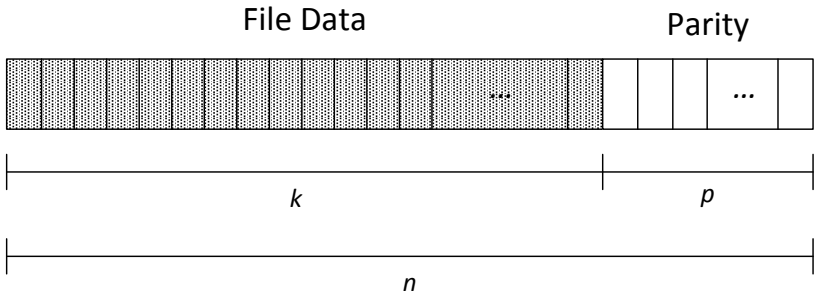

Figure A.1: Reed-Solomon RS(n,k) code

The Reed-Solomon decoding process is able to correct up to $p = (n - k)/2$ data symbols containing errors in a codeword, while in the case of erasures the decoding process can correct up to $p = n - k$ data symbols. Thus,

Reed-Solomon can recover twice as much data symbols when identifying which fragment was lost (erasure).

The required performance to encode and decode RS(n,k) codes is directly proportional to the number of parity symbols generated in a codeword. Thus, a higher or lower p value is translated to a larger or smaller number of errors/erasures that can be corrected and, consequently, more or less CPU processing is consumed.

List of Figures

List of Tables

Acknowledgments

First, I would like to thank God. You gave me power to believe in my passion, strength to progress up to this point, and unconditional persistence belief to pursue my dreams.

I would like to thank my parents, Sezefredo de Almeida Machado and Liana Angelica Sperb Machado, for providing unconditional support throughout my life. I can barely find the words to express all the wisdom, love, and support you have given me. I love you both.

Thanks to my wife, Vanessa de Azevedo Machado, for motivating me in every moment, especially at the final steps. Thanks for not just believing, but knowing that I could accomplish this. As I told you, these are only the first bricks of a large construction which will be built throughout our lives. I love you.

I would like to especially thank Prof. Dr. Burkhard Stiller for believing in my potential and allowing me to be part of the Communication Systems Group (CSG) at the University of Zurich. I will never forget our meeting at the Zurich airport, when we first talked about a possible opportunity to join the group. I am grateful not only for his precious help, feedback, and good will to discuss different subjects, but also for his ability to understand and boost my motivation during difficult moments. Certainly, he has a high degree of influence over the successful completion of this thesis. In addition, I would like to thank Prof. Dr. Filip De Turck for being my co-supervisor and provide me highly valuable feedback in papers, joint projects, and in the final steps of this work. Thank you so much!

I would also like to warmly thank some people that co-authored papers and directly helped on the thesis development. In particular, Dr. Thomas Bocek, Dr. Martin Waldburger, and Dr. Fabio Hecht, which performed a variety of supporting activities, but most importantly reviewed our joint

papers, helped on challenging technical problems, and participated in great brainstorms to reach outstanding solutions.

It has been an immense pleasure working at the CSG, sharing professional responsibilities, as well as enjoying the friendship of my work colleagues, including discussions about diverse subjects during coffee breaks and during events out of the professional environment. Thanks for Christos Tsiaras, Andri Lareida, Patrick Poullie, Radhika Garg, Dr. Corinna Schmitt, Lisa Kristiana, as well as former colleagues Daniel Dönni, Dr. Andrei Vancea, Dr. Hasan, Maurizio Lo Bosco, Peter Ming, Dalibor Peric, Dr. Cristian Morariu, Dr. Peter Racz, and Prof. Dr. David Hausheer.

Thanks for my brothers, Tiago Sperb Machado and Augusto Sperb Machado, for exchanging ideas, opinions, and, despite of the distance, making myself feeling as close to my family as possible. Finally, I would like to thank two persons that are not among us anymore, but considerably collaborated: my grandfather Lydio Sperb, which contributed to achieve my early life steps in a happy way; and my grandmother, Maria de Almeida Machado, which somehow envisioned what would be my professional passion.

Curriculum Vitae

Guilherme Sperb Machado was born on December 22, 1984, in Porto Alegre, Rio Grande do Sul, Brazil. In 1994, his grandmother Maria gave the first computer, an IBM DX2 486 with 4 MByte of RAM, running DOS and Windows 3.11. Late 1996, his parents subscribed to one of the first Internet Service Providers in Porto Alegre city, in order to support school tasks and research. From this point, he started his journey into the Computer Science discipline.

In 2006, he obtained the B.Sc. degree in Computer Science from the Pontifical Catholic University of Rio Grande do Sul (PUCRS, Brazil). His B.Sc. thesis was entitled "White Worm Model – A Conceptual Model of a Worm to Patch Vulnerable Systems". During the time of obtaining his B.Sc. degree he worked as an intern at CPSE (Research Center of Embedded Systems), a research lab funded by Hewlett-Packard R&D Brazil, in Porto Alegre. He was part of the HP Printer's Security team, which had the goal to test network modules and applications of HP printers searching for security-related issues.

In the beginning of 2009 he obtained the M.Sc. degree in Computer Science from the Federal University of Rio Grande do Sul (UFRGS, Brazil). His M.Sc. thesis was entitled "Rollback Support in IT Changes Management Systems", developing a solution to automatically generate rollback plans and perform the appropriate rollback activities if a change plan executed over an IT infrastructure fails. During the time of obtaining his M.Sc. degree he worked in a research project funded by Hewlett-Packard R&D Brazil, having cooperation with Hewlett-Packard Laboratories at Palo Alto (USA) and Bristol (UK). In 2008, he worked as an intern at Hewlett-Packard Labs Bristol, UK, researching topics in the area of IT Change Management, focusing on the generation of remediation plans for IT Change Management systems.

In April 2009 he moved to Zurich, Switzerland, to become a doctoral candidate until mid-2015, as well as a junior researcher in the Communication Systems Group (CSG) at the Department of Informatics of the University of Zurich. He was responsible to a multitude of tasks, including the design, implementation, and prototyping of distributed systems and protocols, the management of the CSG on-production server's infrastructure, development of scripts for system's automation, teaching support to lectures, and the supervision of B.Sc. and M.Sc. students' thesis. Guilherme has been involved in the following research projects: "EMANICS: European Network of Excellence on Management of the Internet and Complex Services", "Accounting and Monitoring of AAI Services (AMAAIS)", "Socially-aware Management of New Overlay Application Traffic combined with Energy Efficiency in the Internet (SmartenIT)", "Management of the Future Internet (FLAMINGO)", and "PiCsMu: Platform-Independent Cloud Storage System for Multiple Usage".

Guilherme's main research interests are in the area of distributed systems and Cloud Computing, in particular Cloud storage, Cloud overlays, distributed file systems, peer-to-peer storage, network protocols, network monitoring, and network security.

His doctoral thesis was supervised by Prof. Dr. Burkhard Stiller (University of Zurich, Switzerland) and Prof. Dr. Filip De Turck (University of Ghent, Belgium).